Gate-Level Superconductor Integrated Circuit Fabrication Process Modelling for Improved Layout Extraction

by

Heinrich Frederick Herbst

Thesis presented in partial fulfilment of the requirements for the degree of Master of Engineering (Electronic) in the Faculty of Engineering at Stellenbosch University

Supervisor: Prof. C.J. Fourie

March 2021

Declaration

By submitting this thesis electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

Date: March 2021

Abstract

Gate-Level Superconductor Integrated Circuit Fabrication Process Modelling for Improved Layout Extraction

H.F. Herbst
Department of Electrical and Electronic Engineering,
University of Stellenbosch,
Private Bag X1, Matieland 7602, South Africa.

Thesis: MEng (Electronic)
March 2021

This thesis presents the development of a fabrication process modelling tool called Katana, created to aid in parameter extraction of superconducting electronic (SCE) integrated circuit (IC) layouts. The program forms part of a toolchain for the development of superconducting technology-based computers. The thesis begins with an overview of the fabrication process of SCE ICs, following which the implementation of Katana is discussed in two chapters. The first chapter on implementation describes a module which generates two-dimensional circuit cross-sections when provided with the mask layout and fabrication process data. The module also assists with manual 3D modelling of circuit elements using strategic slices from the University of Florida's process modelling tool, FLOOXS. The next chapter describes the implementation of an automated, three-dimensional SCE logic-gate modelling module. The module is capable of automatically generating process-modelled circuit representations which can be meshed and fed into finite-element parameter extraction software. The thesis reviews the functionality of the program by generating and analyzing models of Rapid Single Flux Quantum- (RSFQ-)-based superconducting circuit logic gates and confirms the necessity of process modelling through a discussion on how the enhanced models affect parameter extraction. Additionally, the thesis shows that researchers are already using Katana for improved thermal analysis and current density simulation. The concluding section discusses potential further research in future.

Uittreksel

Hek-vlak modellering van supergeleier geïntegreerde stroombaan vervaardigingsprosesse vir verbeterde uitlegonttrekking

("Gate-Level Superconductor Integrated Circuit Fabrication Process Modelling for Improved Layout Extraction")

H.F. Herbst

Departement Elektries en Elektroniese Ingenieurswese,
Universiteit van Stellenbosch,
Privaatsak X1, Matieland 7602, Suid Afrika.

Tesis: MIng (Elektronies)

Maart 2021

Hierdie tesis beskryf die ontwikkeling van 'n vervaardigingsproses-modelleerinstrument genaamd Katana, wat geskep is om met die parameterekstraksie van supergeleier-elektroniese (Eng. SCE) geïntegreerde stroombaanuitlegte (Eng. IC-uitlegte) te help. Die program vorm deel van 'n gereedskapsketting vir die ontwikkeling van supergeleidende-tegnologie-gebaseerde rekenaars. Die tesis bied eers 'n oorsig van die vervaardigingsproses van die stroombane, waarna die implementering van Katana in twee hoofstukke bespreek word. Die eerste implementeringshoofstuk beskryf 'n module wat tweedimensionele stroomdeursnitte genereer wanneer die maskeruitleg en vervaardigingsprosesdata ingevoer word. Die module help ook met die 3D-modellering van stroombaanelemente met die hand, met behulp van strategiese snitte vanuit die Universiteit van Florida se prosesmodelleerder, FLOOXS. Die daaropvolgende hoofstuk beskryf die implementering van 'n geoutomatiseerde, driedimensionele SCE logiesehekmodellering-module. Die module is in staat om prosesgemodelleerde stroombaanvoorstellings outomaties te genereer, wat saamgevoeg en by eindige-element-parameterekstraksiesagteware ingevoer kan word. Die funksionaliteit van die program word ondersoek deur modelle van "rapid single flux quantum"-gebaseerde (RSFQ) supergeleidende-stroombaan- logiese hekke te genereer en te ontleed. 'n Bespreking van die uitwerking van verbeterde modelle op parameterekstraksie bevestig die noodsaaklikheid van prosesmodellering. Daarbenewens word aangetoon dat navorsers Katana reeds vir ver-

beterde termiese ontleding en stroomdigtheidsimulasie gebruik. Die tesis sluit af met 'n bespreking van die potensiaal vir toekomstige verdere navorsing.

Acknowledgements

I would firstly like to thank my mother. Her imparted wisdom has granted me the courage to pursue my dreams; her unyielding support has escorted me towards them. I am sincerely grateful to Professor Coenrad Fourie for providing me with the opportunity to further my skills and assisting me in finding a way forward through complicated problems. I also thank the team at the University of Florida. They created software which I used extensively in my research and were always happy to collaborate. Lastly, I would like to thank the entire QEDA research group. I could always count on my colleagues to aid and inspire me.

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

There are limitations tied to semiconductor technology which are inhibiting the rate at which we can improve the speed and energy efficiency of electronics. Two primary limitations on the clock speed of a central processing unit (CPU) are timing delays and heat build-up [1]. Various electrical constraints cause timing delays [2]. Superconducting electronics (SCE) promises higher clock speeds and lower energy consumption (and thus orders of magnitude less heat generation) than semiconductor electronics.

In 1986, researchers at Moscow State University tested a Rapid Single Flux Quantum (RSFQ) superconducting circuit, successfully operating it at a clock speed of 30 GHz. By 2010 Yokohama National University had demonstrated fundamental RSFQ circuits operating at clock speeds exceeding 100 GHz.

In recent years, due to the potential for use in the intelligence sector, increasing energy-consciousness and reaching limitations on how small semiconductor electronics can be scaled [3], a renewed interest in superconducting electronics has taken root. In 2014, the Intelligence Advanced Research Projects Activity (IARPA) launched the Cryogenic Computer Complexity (C3) program [4] which preceded the SuperTools program launched in February 2018.

The goal of the SuperTools project is to develop a comprehensive set of tools for the design and analysis of SCE circuits, thereby promoting the technology into a state which overcomes the previously mentioned limitations observed in the semiconductor industry [5]. Stellenbosch University is a member of the ColdFlux team which forms part of the SuperTools project [6].

Technology Computer-Aided Design (TCAD) is a well-established subdiscipline of electronic design automation (EDA) which handles the modelling of device fabrication and operation [7]. Superconducting TCAD is still underdeveloped and therefore requires special attention. TCAD proves useful to multiple areas of the toolchain, especially with regards to parameter extraction on a gate-level.

1.2 Background of RSFQ and Modelling

The definition of superconductivity is the phenomenon of the disappearance of electrical resistance below a specific temperature [8]. The temperature below which resistivity disappears is known as the critical temperature or transition temperature. Because niobium is the metal with the highest transition temperature (9.3 K) [8], it makes a prime candidate for fabrication processes.

Most modern computational applications utilise Complementary Metal Oxide Semiconductor (CMOS) technology. Although there are different SCE technologies, we primarily focus on RSFQ technology. However, Adiabatic Quantum Flux Parametron (AQFP) technology also falls within the scope of the ColdFlux team [9]. Superconducting materials require cryogenic temperatures to operate. The SuperTools research effort primarily uses niobium for its fabrication processes [10].

RSFQ technology utilises voltage signal pulses (called magnetic flux quanta or fluxons) to perform logic operations. The required pulses are generated through the manipulation of the current across a circuit element known as the Josephson Junction (JJ).

Because the niobium is in a superconductive state, it is possible to store fluxons indefinitely in superconducting loops. Experiments have indicated that a current in a superconducting coil could persist for years without degradation [11]. A logical "1" is characterised by the presence of a fluxon in a loop within a period defined by timing pulses [12]. A logical "0" is, by expansion, the absence of such a fluxon. These principles allow for the creation of a circuit architecture that operates fundamentally differently to the switching logic levels system of CMOS.

As integrated circuits (ICs) become smaller and denser, it becomes increasingly important to model the physical geometry created by the fabrication process accurately. Parameter extraction directly depends on the integrated circuit's physical geometry. We, therefore, require accurate parameter extraction for accurate circuit simulation. Relevant parameters for extraction include inductance, capacitance and thermal characteristics.

Before C3 and SuperTools, there was no consolidated effort to build a comprehensive toolset capable of very-large-scale integration (VLSI). Previously generations of numerical solvers such as FastHenry [13] were not capable of effectively handling superconducting circuitry. FastHenry created rectangular, uniaxial filaments in order to reach a solution. Solvers required adaptation for the task of handling more complicated geometry. InductEx [14, 15] was created to extract inductance from complex layouts of superconducting circuits. The 3-D numerical engine known as TetraHenry [16] was incorporated into InductEx to aid in this endeavour. TetraHenry is capable of reaching a solution using tetrahedral meshes. Because of this adaptation, in addition to allowing for extraction from more complex geometries, InductEx also became capable of handling complex surfaces, such as those seen in Transmission Electron

Microscopy (TEM) images of fabricated superconducting integrated circuits. Unfortunately, there is no non-proprietary software capable of generating a three-dimensional model with these curved edges.

The University of Florida (UFL) is a member of the SuperTools project and has been working on TCAD software adaptations. There is an ongoing effort to modify the Florida Object-Oriented Device and Process Simulator (FLOOXS) [17] to fit the purposes of the ColdFlux team, most specifically, modelling the device fabrication of a Josephson Junction [18]. The process simulator of FLOOXS, FLOOPS, is capable of generating two-dimensional models of the fabrication processes used in the SuperTools project. There are plans to expand FLOOXS to handle three dimensions eventually.

It has become evident that a bridge is required to connect InductEx's capabilities to the profiles FLOOXS could provide. We require an enhanced 3D modelling system for the project.

1.3 Objectives

In this work, we propose a fabrication process modelling tool, called Katana, that uses strategically simulated two-dimensional contours to create an improved three-dimensional model of a superconducting integrated circuit at the gate level. We use the program to create models of superconducting logic gates to assist in improved parameter extraction. The tool is primarily implemented in C++ and makes use of Python scripting for the three-dimensional model generation phase. We evaluate the applicability of the program in a journal paper by comparing the differences between models generated without process modelling to those generated with Katana. We confirm the effectiveness of the program by testing the usability of Katana-generated models in relevant simulation software. The objectives of this dissertation require the implementation of the following functionality:

1. Enable a designer to peer into the vertical two-dimensional structure of a circuit when provided with a mask layout, cross-sectional co-ordinates and fabrication process specifications.

2. Convert the resulting two-dimensional cross-section representation into a FLOOXS process modelling simulation instruction script.

3. Implement tools which allow a designer to manipulate and combine process-modelled contours to generate more complicated structures in two and three dimensions.

4. Design a system for improved automatic three-dimensional model generation given a circuit layout and fabrication process information.

The concluding item on the above list is the most essential and builds on concepts explored in the preceding tasks. We satisfy the primary objectives by performing the following tasks:

1. An investigation into the background of SCE design.
2. An analysis of fabrication and modelling methods.
3. Implementation of manual modelling mechanisms.
4. Implementation of automatic modelling mechanisms.
5. Documentation of results.
6. Conclusion and recommendations.

In Chapter 2, we discuss the goals of the ColdFlux team and suggest a long-term design flow, describing all the intermediate steps.

Chapter 3 contains a description of the fabrication process at our disposal and elaborates on the fabrication principles used before briefly investigating the Josephson Junction, a critical element of the RSFQ circuit. The importance of parameter extraction is then explained. Once we have established these principles, we investigate FLOOXS as a tool for simulating relevant areas in the fabrication process. We then specify the coordinate system of the project and conclude the chapter with a short description of constructive solid geometry.

Chapter 4 documents the implementation of all the modules and methods which enable Katana to satisfy the first three objectives of the project. The work ranges from interfacing the program with mask layout files to creating cross-sectional representations of a circuit and using them to create advanced models.

We explore the design and implementation of the automated and enhanced three-dimensional model generation system in Chapter 5 with the aim of satisfying the final objective of the project. The enhancement refers to the inclusion of process-modelled features, which are exhibited in the figures of the chapter.

The results chapter, Chapter 6, proves by means of a journal paper that fabrication process modelling has a notable effect on parameter extraction. The chapter also proves that other simulation software can successfully interface with enhanced three-dimensional models automatically created using Katana. We exhibit models of increasing scale and complexity until presenting a model of the OR2T logic gate, the largest cell of the currently used SuperTools RSFQ cell library.

In the concluding Chapter 7, we confirm that all project requirements are satisfied. Furthermore, we discuss the challenges encountered during the research assignment and suggest possibilities for future work. We finish the chapter with a summary of our contribution to the field of SCE.

Chapter 2

Electronic Design Automation of ColdFlux

2.1 EDA in Context

A primary long-term goal of the ColdFlux team is to produce a high-performance 64-bit RISC processor [6,19]. To achieve this goal, we must create a system architecture which translates the circuit requirements as an input, into a working circuit as an output using SCE technology. As with any complicated problem, the first step is to break it down into manageable tasks. When designing new technology, it is easy to fall into the trap of starting entirely from scratch when there are similar solved principles that could apply to the problem. A primary precept of the ColdFlux project is to avoid reinventing the wheel. Fortunately, many fundamental principles translate from semiconductor circuit design to superconducting electronics.

2.2 Hardware Description Language

In order to have a conversation, people must converse in a shared language. One can think of Verilog Hardware Description Language (Verilog HDL) and very high speed integrated circuit (VHDL) as the languages for describing the functionality of a digital circuit. When designing hardware, the use of a HDL is the first step towards solving the initial objectives. The Institute of Electrical and Electronics Engineers (IEEE) adopted the standard for VHDL in 1987. It has since become the industry standard [2].

An ideal superconducting EDA design flow is shown in Fig. 2.1. Designing electronics with a new architecture is a task that requires a certain degree of flexibility. Therefore, the flowchart should be seen as a guideline as opposed to a command.

A designer should convert the concept into a HDL before they can simulate the logic operation of the circuit. They can do this using entities and archi-

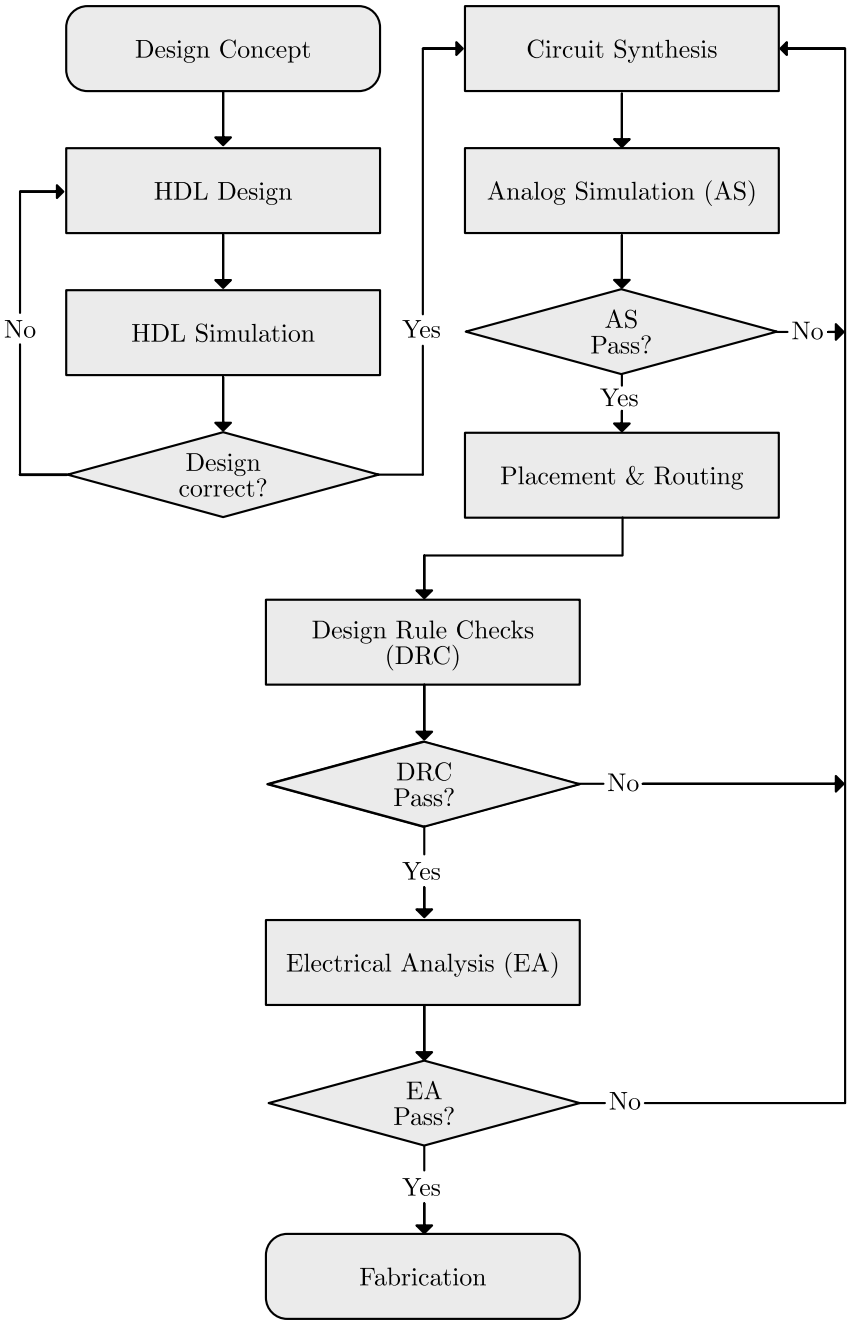


Figure 2.1: Flowchart of ideal Superconducting EDA design flow.

tecture. An entity is an HDL construct which describes a logic function by declaring input and output signals. The architecture describes the relationship between the ports described in the entity. Below is an example of a VHDL and architecture entry:

```
ENTITY OR_AND IS
  PORT( x1, x2, x3 : IN BIT;
        f : OUT BIT );
end OR_AND ;

ARCHITECTURE LogicFunc OF OR_AND IS
BEGIN
  f <= x1 OR (x2 and x3);
end LogicFunc;
```

Listing 2.1: VHDL architecture entry

In Fig. 2.1, HDL design represents modelling the full desired behaviour of the circuit using HDL code. HDL simulation is implemented through discrete event-driven simulation, a simulation method which models the operation of a system as a sequence of events in time. Expressly, with regards to HDL, we accept a circuit condition such as the switching of a logic level, or in our case an SFQ pulse, as an event which must be processed in order to simulate other dependant signals [20].

2.3 Synthesis

If the functionality of the simulated circuit is as desired, the designer can move onto the next step, synthesis. According to the IEEE Standard VHDL Language Reference Manual [21], "A synthesis tool is any tool that interprets VHDL source code as a description of an electronic circuit in accordance with the terms of this standard and derives an alternate description of that circuit". The alternate description of the circuit is often a netlist; a description of the connections of the circuit components which satisfy the design requirements specified in VHDL.

At this stage of the project, there is not yet technology capable of following the ideal design flow as synthesis tools are still in the experimental phase. An example of the design flow for AQFP logic was demonstrated by Yokohama National University (YNU) [9].

A cell library is a collection of circuits which can be used in various combinations to make more complicated circuits. These libraries primarily consist of multiple boolean logic circuits. Examples of these circuits include AND, OR and NOT gates. However, cell libraries can also contain storage functions [22] and technology-specific cells. In order for a synthesis tool to operate, it requires a cell library of the respective technology. Both AQFP and RSFQ logic cell libraries have been developed as part of the ColdFlux deliverables [23]. Some of these cells are discussed in Ch. 6. The RSFQ cell library includes interconnects, boolean logic, buffers and interfacing cells.

2.4 Simulation of Physical Performance

By this stage in the design process, the designer has a configuration of the cells required to solve the specified problem but does not yet know with confidence that the solution will work. Analogue simulation programs such as SPICE (Simulation Program with Integrated Circuit Emphasis) for semiconductor logic [24], or JoSIM [25] for RSFQ logic, simulate the physical operation of a circuit using the supplied netlist and compact SPICE models of the individual circuit elements. The formulation of these SPICE models is discussed in Section 3.11. An example of a JoSIM output is shown in Fig. 2.2. The pulse based logic is evident with voltage spikes lasting less than one tenth of a nanosecond.

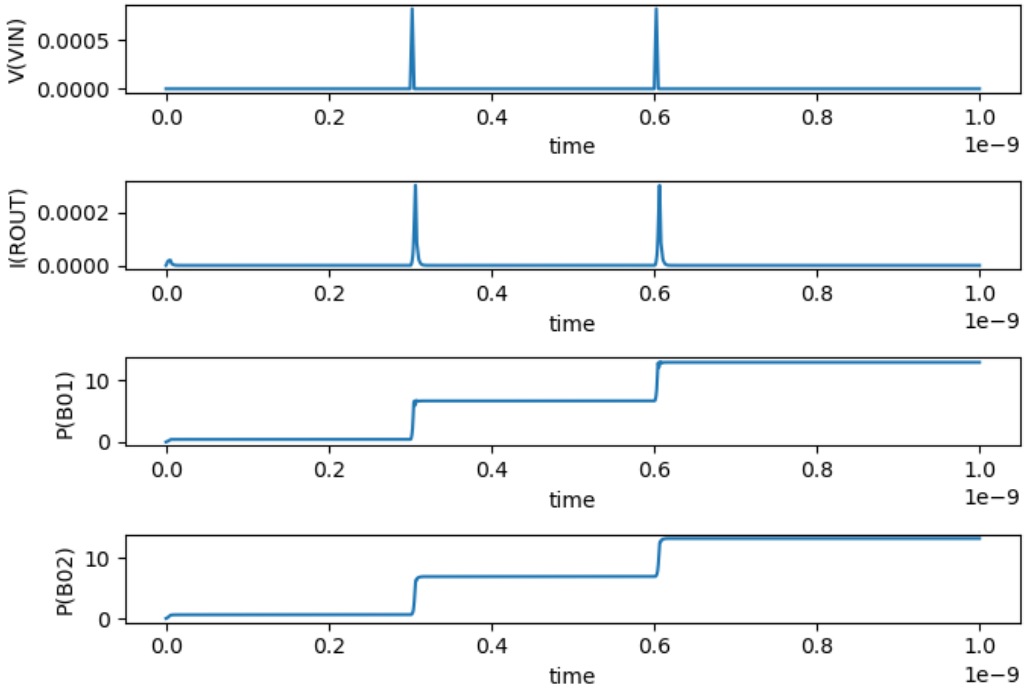


Figure 2.2: Voltage, Current and Phase simulation of Josephson Transmission Line.

2.5 Placement and Routing

Placement and routing are two of the final stages in the EDA design process. Because of the interdependence of placement and routing, most people consider the two steps as one. Placement and routing refer to the process of laying out

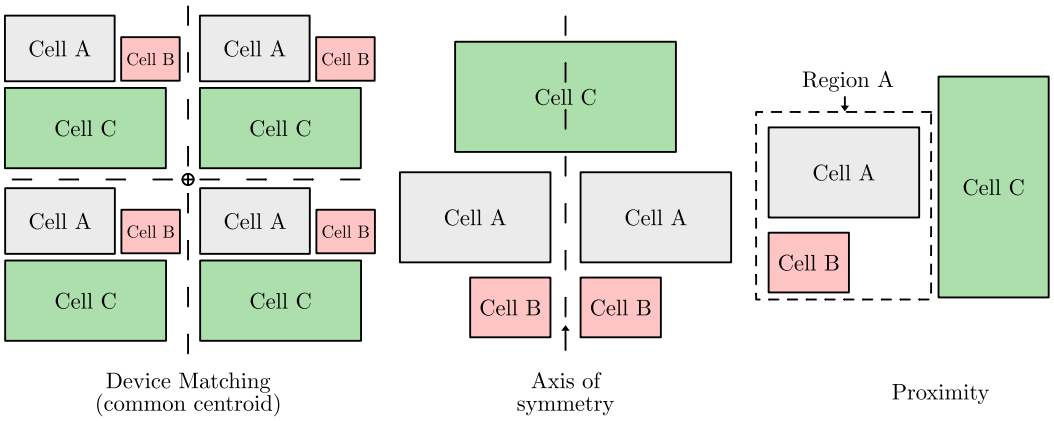


Figure 2.3: Primary topologies of placement and routing.

circuit components within finite space, as well as the process of defining and placing interconnections to achieve a functional hardware circuit layout.

Placement and routing are complicated tasks because multiple objectives must be considered simultaneously in the design. Topology in the context of placement and routing refers to how forming parts are interrelated or arranged. The arrangement of a circuit is critical to its operation.There are three main topological constraints concerning placement and routing of analogue circuits: device symmetry, device matching and device proximity [26]. These topologies are displayed in Fig. 2.3.

Device symmetry refers to the placement of circuit elements along an axis of symmetry, such as the one shown in the middle arrangement of Fig. 2.3. Symmetrical placement helps circumvent geometric and electrical problems.

A parasitic element is an element that has undesirable electrical components such as resistance, inductance and capacitance, which hinder device or circuit operation [27]. Parasitic inductance and capacitance often arise because of the nearness of electrical elements to each other. One of the problems mirrored placement can prevent is the parasitic mismatch between signal flows which should be identical. Symmetry is also useful as a method of reducing timing issues in clock distribution. An example symmetrical clock distribution is topology is the H-tree, illustrated in Fig. 2.4.

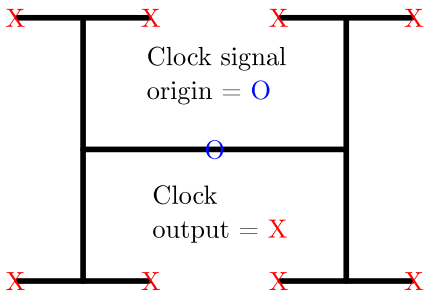


Figure 2.4: H-tree clocking topology.

Device matching is the topological constraint which forces a common centroid or gate orientation, enhancing the benefits of using the symmetrical topology for placement. A common centroid method of device matching is displayed on the left-hand side of Fig. 2.3.

The third topological constraint, device proximity, refers to the requirement of two circuit elements to share the same substrate or exist in the same region such as Cell A and Cell B residing in Region A of Fig. 2.3. An RSFQ specific example would be the requirement of all Josephson Junctions to be placed in the specified region for aluminium substrate deposition.

A designer must be mindful of heat propagation, parasitic effects and the design rules of the fabrication process when working with placement and routing.

2.6 Design Rule Checks

The end product of placement and routing is a physical layout of the geometric patterns required to fabricate the IC. The industry standard for representing this layout is the Graphic Design System (GDS or GDSII) [28] binary file format.

The GDSII format uses a hierarchical system to represent text and planar polygonal information. Usually, the polygonal information is interpreted as a mask or lack thereof, depending on how the fabrication process defines the layer number. Because GDSII is a planar file format, further information about the fabrication process is required. This topic is further discussed in Section 4.2.

The GDSII file must undergo Design Rule Checking (DRC) before fabrication. DRC is the process of verifying that no rules specified by the fabrication process are broken in the layout file. Design rules are created to ensure that the fabrication process is capable of successfully producing the IC and ensures that it operates as intended. If a designer ignores the design rules, they might unwittingly introduce undesired features such as short circuits and other deformities [29]. Two examples of a design rule for a hypothetical process would be:

- The local density of any 300x300 µm region on a metal layer must be between 20% and 80%.
- The minimum width of a through-silicon via (TSV) must be 0.7 µm.

Design rules are usually implemented due to the limitations of the machinery used to fabricate the IC but could also be from experience of previous fabrication attempts.

2.7 Electrical Analysis

In addition to satisfying design rules, the designer must ensure that electrical rules are not violated in the final layout. The physical layout should be analysed by simulation software that, among other things, could:

- Excite the circuit model with current.
- Determine if critical current is exceeded unintentionally.
- Determine if critical temperature is exceeded unintentionally.
- Identify unexpected mutual capacitance and inductance.
- Verify that there are no timing violations.

A robust toolchain, once fully implemented, should leave very few issues to be identified at this stage, as the previous stages could be designed to minimise design, electrical and timing violations. However, with new technology, it is likely that at least in the early stages of development, this phase will be critical to identifying errors in designs.

2.8 Fabrication

Once the designers are as confident as they can be, within the limitations of design and simulation tools, they can move to the next phase: Fabrication. The fabrication process is discussed in detail in Chapter 3.

Chapter 3

Fabrication Process: Overview and Modelling

3.1 Fabrication in the Context of RSFQ

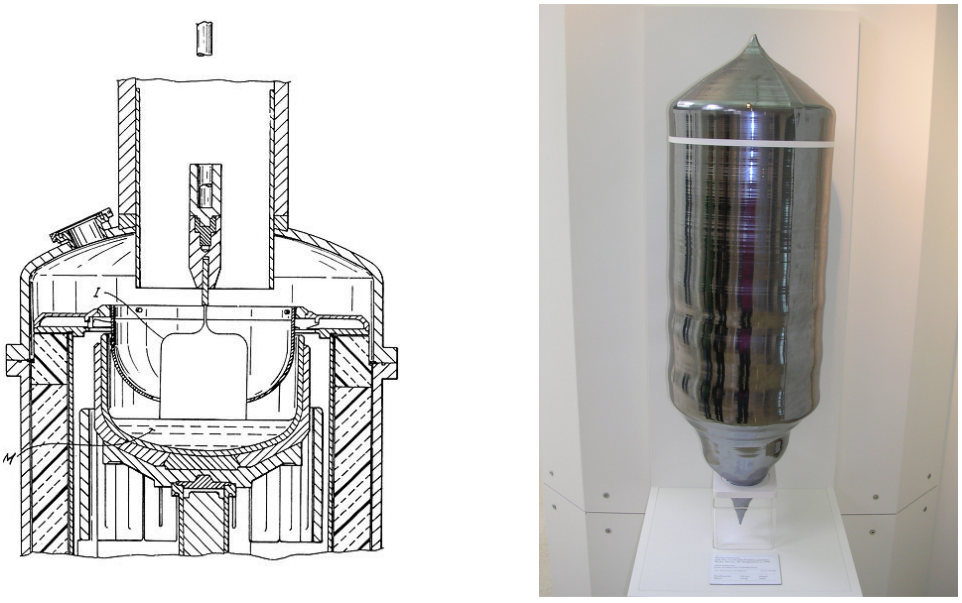
The fabrication process of integrated circuits varies depending on the type of circuitry required. Regardless of functionality, most modern circuits are made on silicon substrates [30]. As members of the ColdFlux project, we design our cells around the Massachusetts Institute of Technology Lincoln Laboratory (MIT LL) SFQ5ee fabrication process. In this section, we discuss the fabrication process techniques which are relevant to RSFQ development.

Before the integrated circuit fabrication process can begin, the silicon wafer must be created. The wafer acts as the foundation for the circuit. The SFQ5ee process makes use of a 200 mm wafer foundry [10].

3.2 Wafer Preparation

The industry-standard method of making silicon wafers is to use the Czochralski growth method, named after its inventor, Jan Czochralski [30]. The method requires electronic grade polysilicon, which is the product of extensive refinement of quartzite, a rock constituting primarily of silicon dioxide. After refinement, the polysilicon is so pure that less than 0.0000000001% of the final product is impurities [30]. The growth process takes place in a Czochralski growth system which compromises of a furnace inside a chamber. A Czochralsi furnace is displayed in Fig. 3.1a.

The first step of the process is to insert the polysilicon into the crucible. An inert gas such as argon displaces and consequently replaces air in the chamber. The polysilicon must be heated to approximately 1500 °C to reach a molten state. Once the melt is ready, a seed crystal is carefully depressed into it to act as a catalyst for crystallisation. The crystal is simultaneously rotated and raised out of the melt, causing the silicon to cool and crystallise with the same



(a) Czochralski furnace [31]. (b) Silicon boule [32].

Figure 3.1: Czochralski furnace and silicon boule.

orientation as the seed crystal. The larger crystal produced is called a boule, shown in Fig. 3.1b.

The boule is characterised according to the quality of the crystal and its resistivity property. The main body is ground to a diameter which is slightly larger than the required wafer diameter after the seed and tail are severed. The boule is then submerged in a chemical etchant to remove defects caused by the grinding [30].

High-grade steel wires impregnated with diamond particles cut the boule into multiple wafers which will next undergo lapping, the process of precisely rubbing two surfaces together to produce a smooth surface. After lapping, the wafer is etched and finally polished in an electrochemical process [30] to produce the final product ready for the next stages of fabrication. The fabrication processes used in the SFQ5ee process are illustrated in Fig. 3.2.

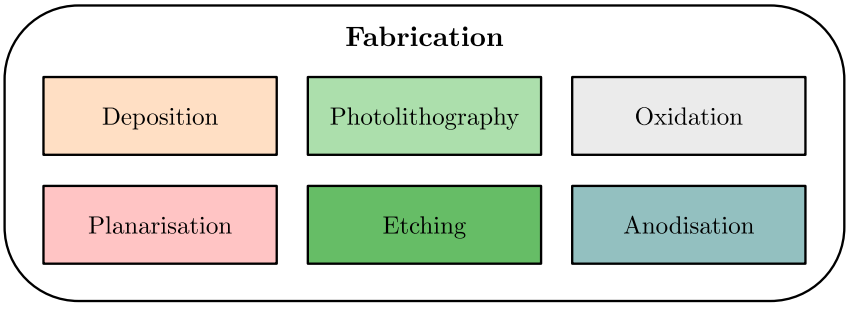


Figure 3.2: SFQ5ee fabrication process steps.

3.3 Deposition

The SFQ5ee process makes use of DC magnetron sputtering for deposition of Niobium layers and plasma-enhanced chemical vapour deposition (PECVD) for the deposition of the interlayer dielectric of SiO_2 [10]. A diagram of magnetron sputtering is shown in Fig. 3.3.

The following three steps initiate magnetron sputtering:

- Evacuate the chamber.
- Supply the vacuum chamber with argon gas.
- Create a high potential difference in the system.

This process ionises the argon gas. Argon ions are attracted to the cathode and collide with the target material, which, four our purposes, is niobium. This process is known as ion bombardment. The electrons which have broken free from the argon ions remain trapped in the magnetic field, increasing the ion density and bombardment rate [30].

In some processes, ion bombardment is used intentionally for the purpose of etching. Here, the niobium is being etched away from the target. However, etching is not the prime goal of this process. The collision of the argon cations

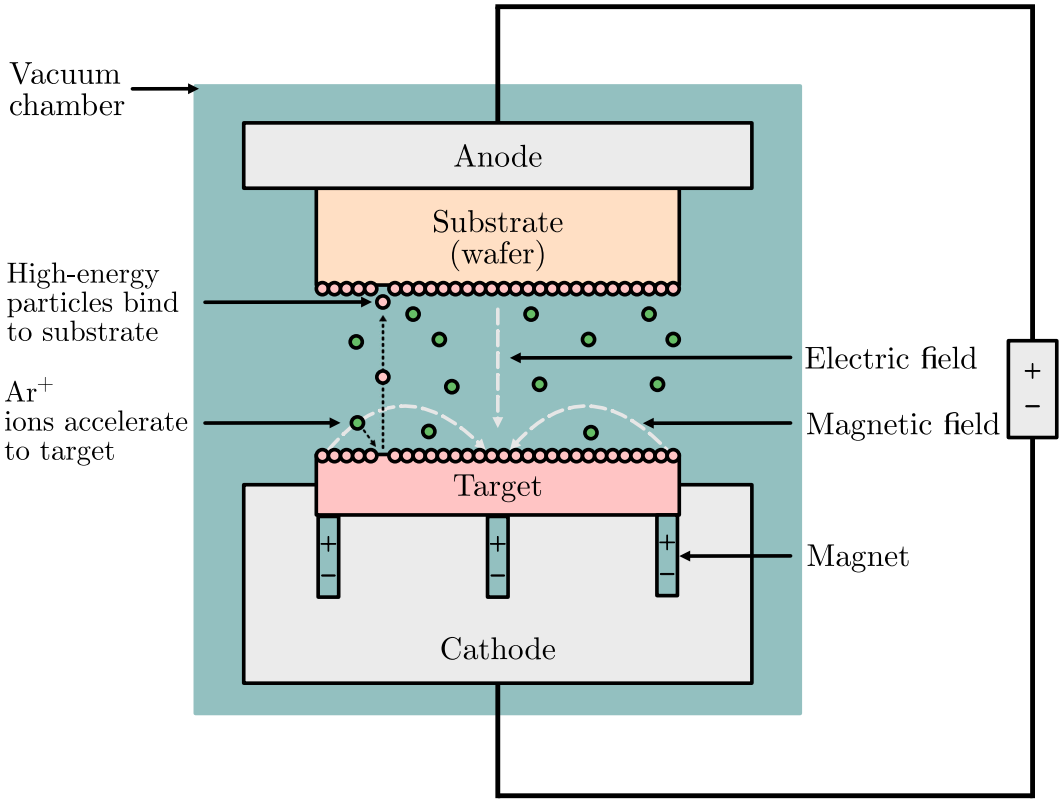


Figure 3.3: Magnetron sputtering diagram.

with the target surface supply some niobium particles with enough energy to break away and deposit on the substrate [30], allowing manufacturers to deposit thin films of material such as the niobium which is required to fabricate superconducting ICs.

Magnetron sputtering is not viable for deposition of silicon for several reasons. Therefore, a different system must be used. The favoured industrial system for large wafers, such as the 200 mm wafers created in the SFQ5EE process, is parallel-plate, hot wall PECVD. In order to understand the process, we will first discuss a very simple thermal CVD reactor, illustrated in Fig. 3.5. The primary steps of the process are shown in Fig. 3.4.

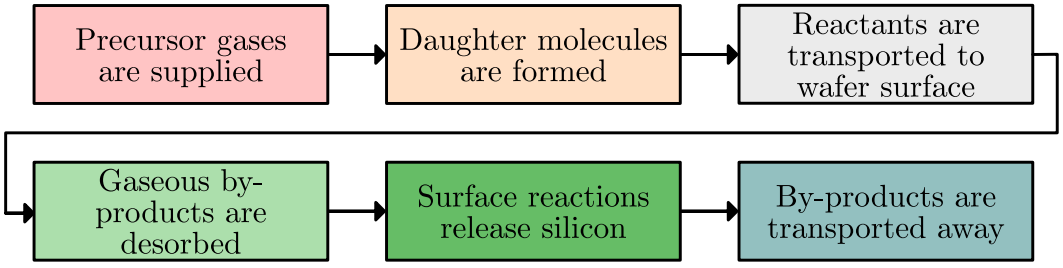


Figure 3.4: Basic chemical vapour deposition process.

The driving ingredient of the PVD reaction is silane (SiH_4), a hazardous and toxic but also beneficial gas. Because of how easily silane reacts with oxygen, it is classified as pyrophoric, meaning that the gas ignites spontaneously in air at or below 54 °C. Silane is used in electronics manufacturing because it is a precursor to elemental silicon. Silane is mixed with a carrier gas such as hydrogen to improve the uniformity of deposition across the wafer. The following overall reaction occurs in PVD:

$$SiH_4(g) \longrightarrow Si(s) + 2\,H_2(g). \tag{3.1}$$

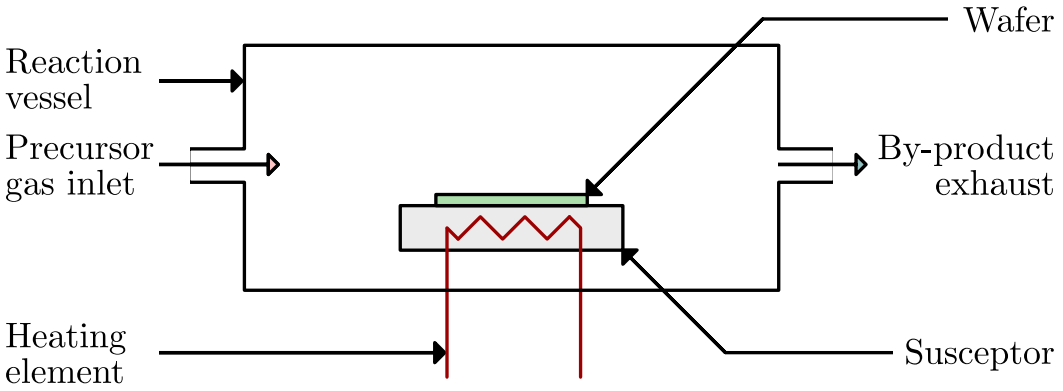


Figure 3.5: Simplified PVD reactor.

Equation 3.1 is a simplification of multiple reactions that take place in the chamber [30]. Three of the most occurring reactions will be discussed: The first reaction,

$$\mathrm{SiH_4(g)} \rightleftharpoons \mathrm{SiH_2(g)} + 2\,\mathrm{H_2(g)}\,, \tag{3.2}$$

is homogeneous, meaning that the reaction occurs spontaneously in the enclosed mixture of gases. The letters in parentheses indicate (g) for gaseous and (s) for solid phases. This reaction is reversible and occurs in both directions inside the system. An equilibrium of the two gases is reached in the reaction vessel. Silyene ($\mathrm{SiH_2}$) is one of the daughter molecules which are formed in step 2 of Fig. 3.4. The silyene molecules circulate in the chamber and some of them come into contact with the wafer. The wafer, heated by the susceptor, is hot enough to facilitate the adsorbtion of the silyene:

$$\mathrm{SiH_2(g)} \rightleftharpoons \mathrm{SiH_2(a)}\,. \tag{3.3}$$

The marker in parentheses (a) indicates an adsorbed compound. An adsorbed compound is one which is adhered to a surface. Gaseous byproducts are desorbed in step 4 of Fig. 3.4:

$$\mathrm{SiH_2(a)} \rightleftharpoons \mathrm{Si(s)} + 2\,\mathrm{H_2(g)}\,. \tag{3.4}$$

Traditional thermal CVD methods, such as those used in the reaction system described above, are not viable for the SFQ5ee process because thermal CVD requires an operating temperature range between 650-750 °C [30]. These temperatures would make the substrate too hot for the aluminium layer which would begin to melt at 660.3 °C [33]. Additionally, thermal CVD is not capable of filling smaller features to the level of efficacy required for our superconducting ICs.

Plasma-enhanced chemical vapour deposition accommodates the requirement of lower substrate temperatures by using radiofrequency (RF) plasma to provide energy for the chemical reactions. Ions bombard the surface of the wafer, transferring enough energy for (3.4) to take place. Smaller features are also better filled because of the low temperature and uniform distribution of bombardment. The SFQ5ee process involves depositing $\mathrm{SiO_2}$ using low pressure chemical vapor deposition (LPCVD) or PECVD. A schematic diagram of a PECVD system is shown in Fig. 3.6. The showerhead aids in equal distribution of the precursor gas into the plasma field.

3.4 Oxidation

Oxidation is defined as a reaction where electrons escape from an atom, molecule or ion. The process received its name because oxygen was the first discovered oxidising agent. In the SFQ5ee process, an aluminium layer is deposited

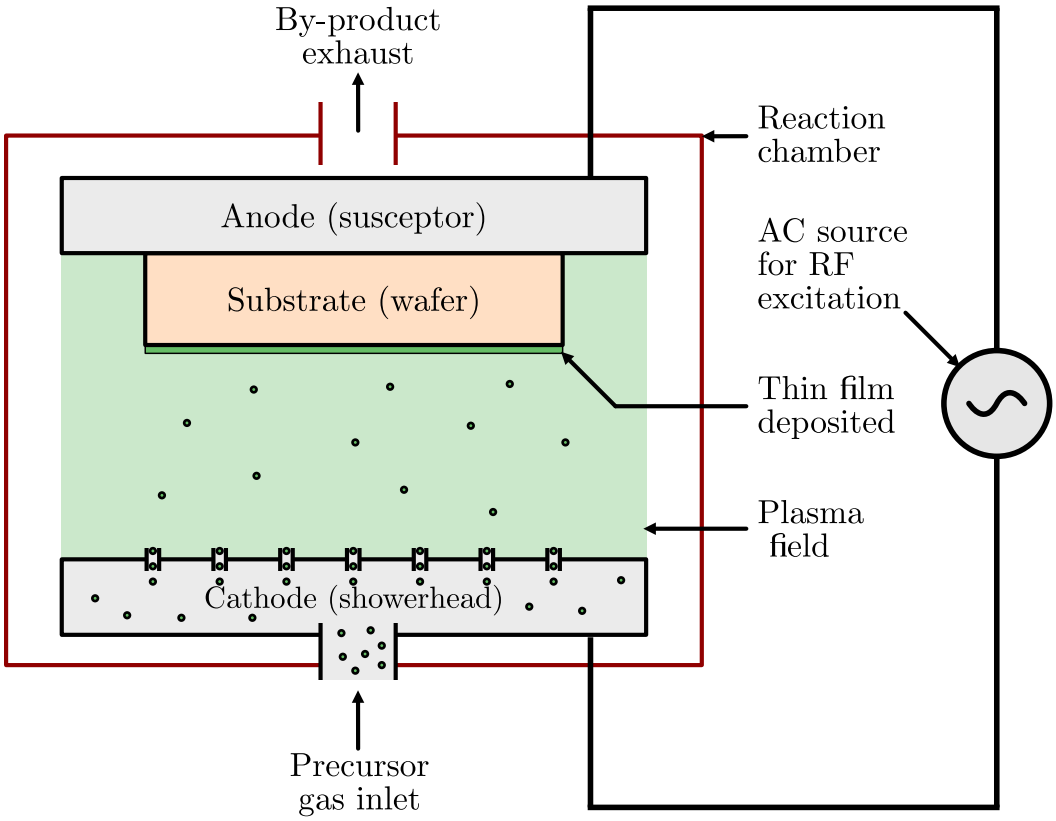


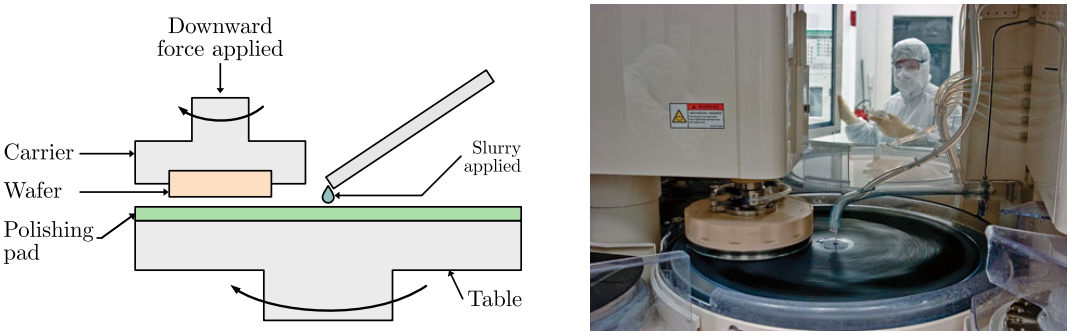
Figure 3.6: PECVD schematic.

through magnetron sputtering. It then undergoes thermal oxidation when exposed to pure oxygen gas. The equation for oxidation of aluminum is

$$4\,\mathrm{Al(s)} + 3\,\mathrm{O_2(g)} \longrightarrow 2\,\mathrm{Al_2O_3(s)}\,. \tag{3.5}$$

3.5 Planarisation

Planarisation, also known as chemical mechanical polishing (CMP), is the process of removing features across a surface in order to create a smooth, flat plane. Planarisation is an essential step in multilayer IC fabrication. It is primarily important because of the need to bring lithographic patterning steps down to smaller dimensions [34]. The process ensures the next layer deposited on the wafer will have an even foundation on which to rest. Without CMP, stacking of layers would not be viable beyond a certain thickness. Each subsequent layer deposited would be strongly affected by the topography of all previous layers. Planarisation also ensures consistency in the thickness of insulating layers. The process is achieved through a combination of chemical and mechanical methods. A basic diagram of the CMP process is illustrated in Fig. 3.7a. A CMP machine is shown in Fig. 3.7b.



(a) CMP schematic.

(b) CMP machine [35].

Figure 3.7: Schematic and picture of planarisation machine.

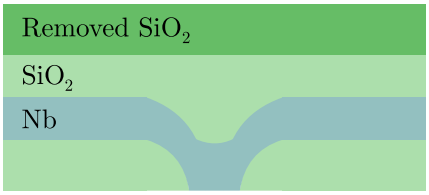


Figure 3.8: Planarisation over a niobium-filled hole.

The most common method used for CMP is to deposit a thick layer of SiO_2 and then polish the wafer in a slurry of abrasive SiO_2 particles called colloidal silica. The slurry contains an etching agent such as hydrogen fluoride (HF) to aid in the polishing [30].

An example of where the benefit of CMP can clearly be observed is when a material is deposited into etched holes of the previous layer, such as in Fig. 3.8. The niobium layer would sag into the hole, resulting in a sagging layer of isolation above it. However, because the isolation layer is grown beyond its final thickness and then planarised down, the distortion is completely mitigated.

3.6 Photolithography

Ion bombardment as a method of etching has been discussed briefly as part of Section 3.3. Such etching technologies generally target whatever material is exposed. There are processes which are selective, meaning that only specific types of materials are etched away. However, even with selective processes, one might not wish to remove all the material across a layer. Therefore, a manufacturer needs to be able to choose where on a layer to etch and where not to. This ability to choose is achieved through photolithography, a method to cover material in some areas and expose it through others.

Photolithography is used to form masks which cover IC wafers. Masks were briefly mentioned in Section 2.6. Photomasks act as the link between fabrication works and the chip designer. The basic principle is to match a

mask layout file such as GDSII to a real-life equivalent to use in fabrication. Each mask is useful for one specific layer of the process. Photomasks require a very flat surface to be effective.

Lithography is considered the most intricate, costly, and crucial process in microelectronic fabrication, accounting for on average one third of all fabrication costs. A conventional silicon technology requires approximately 15-20 different masks [30]. The SFQ5ee process uses deep-UV photolithography for all process layers [10]. A schematic of a projection imaging system [36] is shown in Fig. 3.9. The combination of optical source and condenser lense is called an illumination system.

After SiO_2 has been deposited onto the wafer, a photoresist adhesive is applied. The most common adhesive agent used is hexamethyl disilizane (HMDS). Photoresist, a light-sensitive polymer is spin coated over the wafer and subsequently baked [36].

Photoresist is intended to resist etching, but only in specific areas. The condenser lens focuses light onto the mask. The pattern caused by the mask is then refracted over the desired mask surface. Wherever positive photoresist is exposed to light, it undergoes a chemical reaction which makes it soluble. These soluble areas are subsequently washed away to expose the underlying area for etching. This process is illustrated in Fig. 3.10. The remaining photoresist is removed from the wafer using liquid resist stripper after etching is complete.

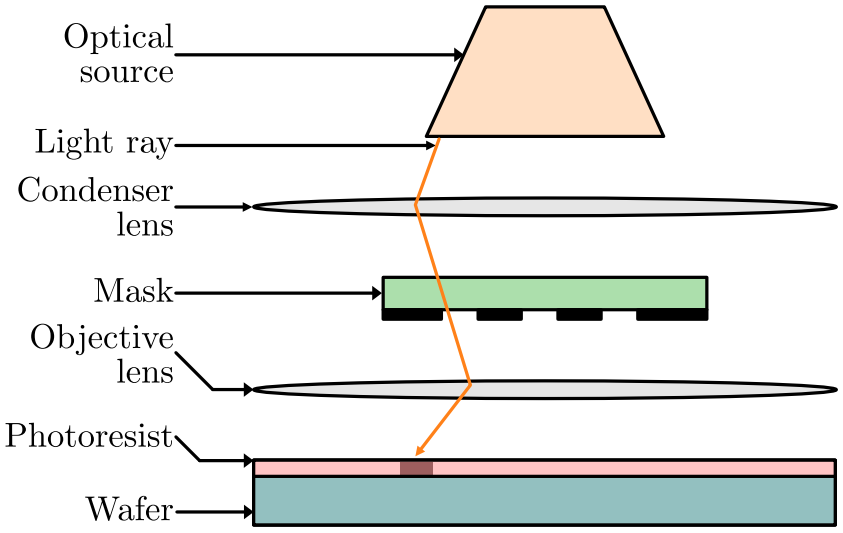


Figure 3.9: Projection imaging system schematic.

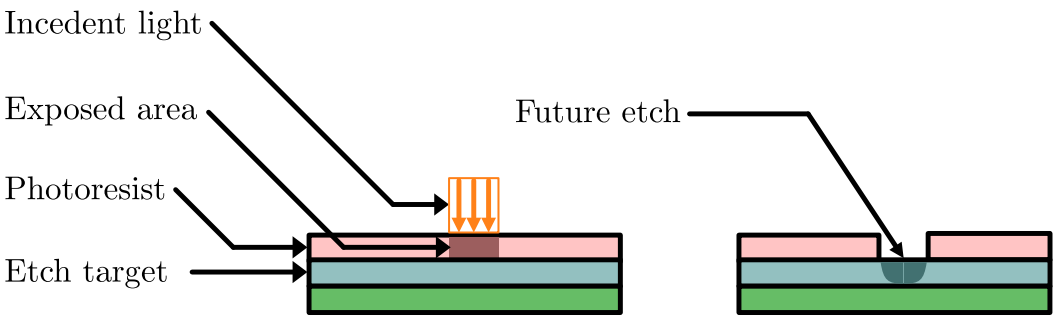


Figure 3.10: Photoresist removal.

3.7 Etching

High-density plasma etching (HDPE) is used in SFQ5ee fabrication. In Section 3.3, we looked at deposition using PECVD. The method of HDPE is nearly the same as that shown in Fig. 3.3. However, because etching is the primary goal of this step, there is no target material at the cathode.

In HDPE, a resultant etch is both a combination of chemical etching from the reaction of the etching target film with plasma, and physical sputtering from the directional bombardment of ions colliding with the wafer.

It should be noted that ions from the high-density plasma bombard both the photoresist and the exposed etching target. However, the process has excellent selectivity, meaning that in a given period the plasma removes far more of the etch target than it does the protective photoresist coating.

HDPE is a form of dry-etching because no etchant solution is required. We refer back to Fig. 3.8 for an example of an etch. The middle of the lowest layer of SiO_2 is etched away before niobium is sputter-deposited over it.

3.8 Anodisation

Anodisation is an electrolytic process where a target metal is used as the anode and oxidised, thus coating the metal surface [37]. Anodisation is generally used as a method of increasing corrosion resistance. The SFQ5ee process includes an anodisation step for the counter electrode and Nb/Al/AlOx layers of the Josephson Junction. An ammonium pentaborate solution in ethylene glycol is used as the anodising agent. Anodisation affects the electrical parameters of the thin Nb/Al/AlOx tunnelling barrier [38].

3.9 End Result of Fabrication

Fig. 3.11 is an adapted drawing of a scanning electron microscope (SEM) image of an SFQ5ee circuit cross-section [10]. This is how the end result looks once all the process steps are completed. The contact pads which allow for interfacing

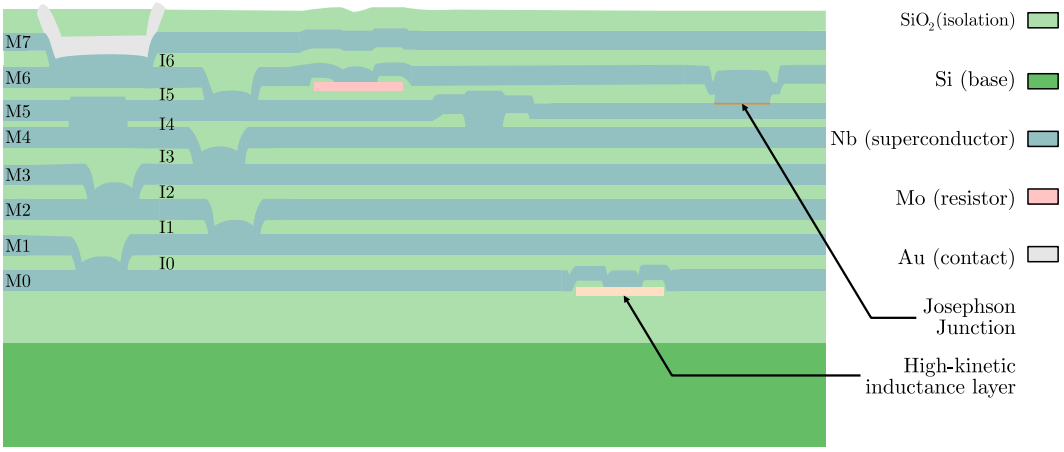


Figure 3.11: SFQ5ee process cross-section.

with the chip are made of gold or platinum. A molybdenum resistor layer is included between isolation layer 5 and metal layer 6. The largest thickness of any metal layer in the process is 200 nm.

3.10 The Josephson Junction

The Josephson Junction is the central device for RSFQ functionality. Over the years, there have been many different kinds of Josephson junctions. The common denominator between all the different sizes, orientations and material choices is the presence of a thin layer of non-superconducting metal or an insulating layer deposited between two layers of superconducting material. The thin layer is called a weak link.

Brian Josephson originally proposed that electron pairs in superconductors can pass, or "tunnel" through the weak link. He took this hypothesis even further by figuring out the equations relating current to voltage in the junction. He eventually won the Nobel prize in physics for these discoveries [39]. The first Josephson equation,

$$I_J = I_c \sin(\varphi(t)) \,, \tag{3.6}$$

describes the current through the junction as a function of the junction's critical current, I_c, and the phase difference between the two wave functions. The symbol, $\varphi(t)$, is the difference between the phases of the first superconductor ϕ_1, and the second, ϕ_2. The critical current is a function of the shape and structure of the Josephson Junction [40]. Therefore it is a parameter which can be extracted from TCAD models or determined experimentally.

Josephson's second equation,

$$V(t) = \frac{\Phi_0}{2\pi} \frac{\partial \varphi(t)}{\partial t} \,, \tag{3.7}$$

relates the voltage across the junction to the difference between phases. In the latter equation, Φ_0 is the magnetic flux quantum, also known as a fluxon. The magnetic flux quantum is the mathematical inverse of the Josephson constant K_J, where e and h represent electron charge and Planck's constant.

$$K_J = \frac{2e}{h} \tag{3.8}$$

The SFQ5ee process makes use of Nb/AlOx/Nb Josephson junctions. This type of junction is called a superconductor-normal metal-superconductor (SNS) JJ. The weak link is only 9 nm thick, compared to the 459 nm combined thickness of the base electrode, counter electrode and weak link. The thickness ratio of N to SNS is, therefore, 1 to 51. A Gmsh hand-modelled example of a JJ is shown in Fig. 3.12. An illustration of a Josephson junction cross-section adapted from an SEM image [41] is shown in Fig. 3.13. In Section 3.11 we discuss the extraction of model parameters. A well represented Josephson junction model is important when calculating equivalent compact SPICE models, one of the objectives of a study in Section 6.8, so it is important to understand how they are fabricated. The element geometry provides a hint as to how a designer could begin to model the junction in three-dimensions. The JJs fabricated in the SFQ5ee process are all circular, therefore an approximated 3D model could be created by sweeping a single two-dimensional section 360 degrees about the vertical axis.

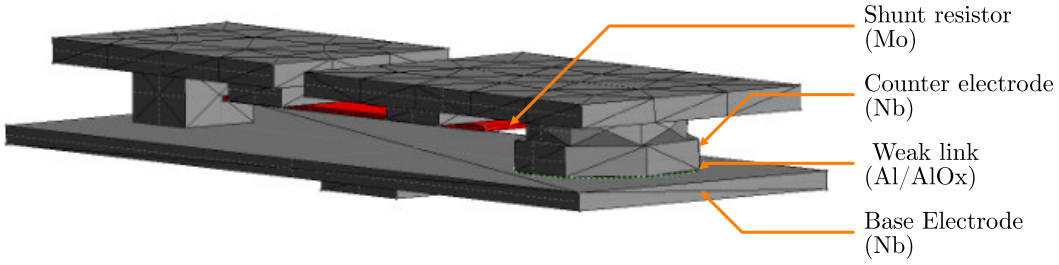


Figure 3.12: Josephson junction with shunt resistor.

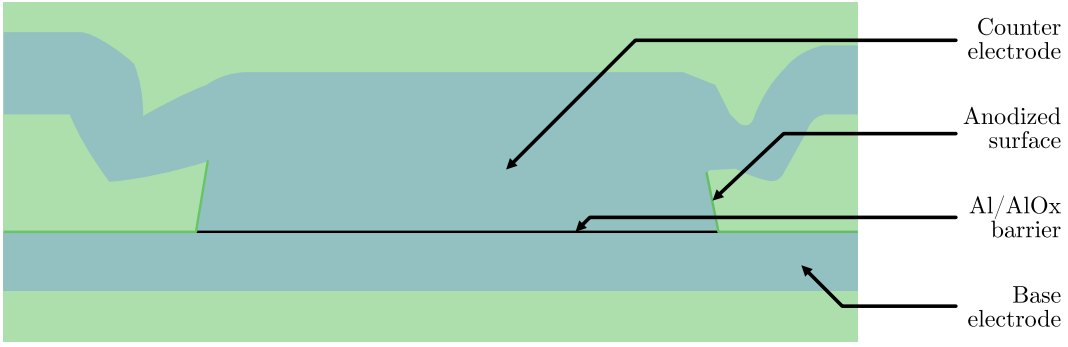


Figure 3.13: SNS JJ cross-section.

3.11 The Importance of Parameter Extraction

We define parameter extraction as the calculation of electrical or physical properties of a circuit or circuit element based off of a geometrical model of said circuit or circuit element. There are many relevant properties which could be determined from a geometric model in the context of EDA. Electrical properties are of particular importance to SPICE simulation. Therefore we require methods to extract parameters so that we may correctly and accurately imitate circuit functionality before fabrication.

Three classic parameters of relevance to electronics are resistance, inductance and capacitance. For now, we will use resistance as an example so that we may convey how a physical model can significantly impact a simulation.

Electrical resistance is defined as the opposition to current flow in an electric circuit. We can calculate the resistance of a conductor using,

$$R = \frac{\rho L}{A}\,, \tag{3.9}$$

where ρ represents the electrical resistivity of a material, L represents the length and A, the cross-sectional area. Resistivity is dependent on the temperature of the material and the type of material. This dependence explains why the resistance of an element increases as it heats up. Resistance can also be calculated experimentally using Ohm's law:

$$R = \frac{V}{I}\,. \tag{3.10}$$

The letters V and I represent the instantaneous potential difference and current across the element, respectively.

We now consider a hypothetical nano-scale molybdenum resistor with the properties of Fig. 3.14. For our argument, we assume that the designer models the resistor as a rectangular prism with the parameters shown in Fig. 3.14a. If, after fabrication which made use of etching, the resistor had the dimensions

(a) Rectangular resistor model. (b) Trapezoidal resistor model.

Figure 3.14: Cross-sections of hypothetical resistor models.

of Fig. 3.14b, then the resistance of wires of identical lengths and operating temperatures would differ by a factor of $A_{(a)}/A_{(b)}$. The model-extracted resistance values would be 25% higher than they should have been. While this is an exaggerated scenario, it shows why it is important to use a model as close to the fabricated circuit as possible. In Chapter 6, we investigate the effect of different geometry on a passive transmission line (PTL). Parameters of importance in the investigation are:

- characteristic impedance Z_0
- complex capacitance $\hat{C}$
- complex inductance $\hat{L}$
- propagation constant γ
- maximum current density I_z

All of the above parameters were extracted from a mesh of a 2D cross-section of approximated and predicted geometries. A 3D model has more intricacies and more areas where parameter extraction might vary based on the approach to geometrical modelling.

Another point of consideration is that circuit geometry affects the rate of heat conduction, which is a function of the material's properties and its physical dimensions. We can see this relationship with Fourier's law of heat conduction in one-dimension [33]:

$$\dot{Q}_{cond} = -kA\frac{dT}{dx}\,. \tag{3.11}$$

The rate of heat conduction, $\dot{Q}_{cond}$ is measured in watts. The letter k represents the thermal conductivity of the material. The cross-sectional area, A, and temperature gradient, dT/dx, are both geometry-dependant. While this is a straightforward case of heat conduction, it conveys the basic principles of heat transfer. Heat transfer is an important consideration in superconducting electronics. A superconducting material loses its superconductivity and becomes a resistive element anywhere that the critical temperature T_c of the superconductor is exceeded. This loss of superconductivity could potentially ruin circuit operation.

3.12 The Florida Object-Oriented Process Simulator

Over the years in which semiconductor technology has matured, engineers have developed fabrication process modelling software to extract the most accurate

parameters from devices. The Florida Object-Oriented Device, Process and Reliability Simulator (FLOOXS) is one such effort.

SUPREM-IV, FLOOXS's predecessor, was released in 1986 by Mark Law, Connor Rafferty and Robert Dutton at Stanford University. The authors developed the software with funding from the Defense Advanced Research Projects Agency (DARPA) for modelling the fabrication of semiconductor devices, as well as determining their electrical properties after undergoing processes such as doping and oxidation. FLOOXS continued to improve along the same path. In 2017 the University of Florida joined the SuperTools project to begin adapting FLOOXS for superconducting electronics. Their primary task is to bring FLOOXS to a stage where it can model the fabrication of a Josephson junction accurately, taking all relevant physical effects into account for the most accurate parameter extraction.

The version of FLOOXS [42], which we use for this study, supports two-dimensional simulation of the following process steps:

- Isotropic and sputter deposition
- Isotropic and anisotropic etching
- Chemical Mechanical Polishing
- Oxidation/Anodization

The process simulator of FLOOXS, FLOOPS, achieves simulation through the utilization of numerical methods such as the finite element method and the level set method (LSM) [17, 42]. The LSM tracks the movement of interfaces implicitly. After each process step, FLOOXS automatically meshes the contents below the interface using Triangle [43] or Gmsh [44].

3.13 Computer Graphics

The majority of computer graphics programs represent geometry using Cartesian coordinates. There are two primary systems used [45]: The left-hand coordinate system shown in Fig. 3.15a and the right-hand, positive z-axis coordinate system is shown in Fig. 3.15b. Because we use Gmsh, FreeCAD and Clipper [46], we have chosen to adopt their convention and use the right-hand, positive Z-axis coordinate system.

In Gmsh and Freecad, points are represented by their x, y and z co-ordinate values. Lines are represented as lists of point values with the start point specified first and the end point last. Therefore, the lines have a specified direction. Klayout and Clipper use the same system but without the Z axis as both are two-dimensional.

(a) Left-handed coordinates. (b) Right-handed coordinates.

Figure 3.15: Cartesian coordinate systems.

3.14 Constructive Solid Geometry and Surface Modelling

In Chapter 4, we make use of points, lines, surfaces and transformations of the aforementioned in Gmsh to form a complex shape from a shell. The aim is to hand-model superconducting circuits using the features of Katana specified in the chapter. During the later phases of program development, we discovered the shortfalls of these methods and began to investigate alternatives which better matched the scale of the project. The automatic layout generation module of Katana, discussed in Chapter 5, utilizes the FreeCAD application programming interface (API) for model generation.

FreeCAD [47] is an open-source, general purpose 3D CAD program with Python scripting support. The program is cross-platform and has an active community. FreeCAD makes use of the OpenCascade CAD engine, the same engine which Gmsh extensively supports. We decided to incorporate FreeCAD into the workflow of this project for all of the above reasons.

One of the primary ways a designer can create parts in FreeCAD is through the use of constructive solid geometry (CSG). CSG is a paradigm of CAD modelling which represents solid shapes through a sequence of Boolean operations on simpler shapes, called primitives [48]. Fig. 3.16 shows a modified [49] example of a CSG workflow.

FreeCAD supports three primary boolean operations:

- Union
- Difference
- Intersection

The Boolean union function fuses two primitives together. The difference function subtracts the overlapping volume of the second primitive from the first. The intersection retains only the overlapping volume between two primitives.

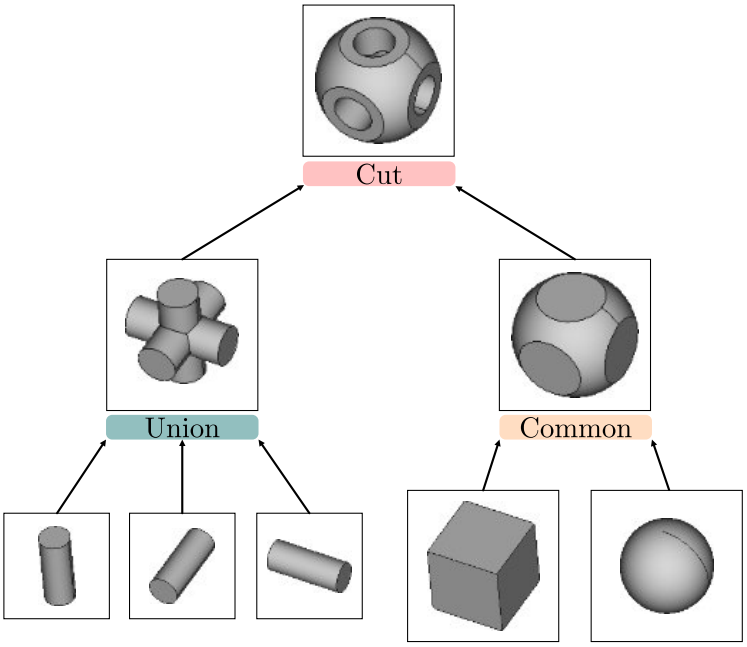


Figure 3.16: Constructive solid geometry workflow.

Chapter 4

Implementation: Manual Modelling Methods

4.1 Implementation Overview

We refer back to Section 1.3, where we discuss the primary objectives of our work. This chapter details the implementation of the first three objectives in Katana, starting with: Enable a designer to peer into the vertical two-dimensional structure of a circuit when provided with a mask layout, cross-sectional co-ordinates and fabrication process specifications.

Before we can develop a program for analysing the two-dimensional cross-sections, we first have to create an interface between our program and the GDSII mask layout file. Our solution to this challenge is the creation of a C++ GDSII file manipulation library, titled GDSCpp [50]. Section 4.2 describes this program's implementation and functionality.

After the completion of GDSCpp, we implement cross-section generation from a mask layout and specified process information. We describe cross-section generation in Section 4.3.

From the cross-section information, we create a script generator for FLOOXS so that we can generate two-dimensional process-modelled meshes of our cross-sections more easily. The FLOOXS script generation covers our second primary project objective: Create a module which translates said cross-section to a FLOOXS input. We discuss FLOOXS script generation in Section 4.4.

We aimed to take the FLOOXS output of strategic slices and combine them into three-dimensional models of circuit elements such as the Josephson Junction and PTL. However, at the time, FLOOXS only exported meshes of the simulation.

Meshes are ordinarily the end-product of the modelling process, usually used solely for finite element simulation. Meshing is a complicated process. Therefore, one cannot merely duplicate, rotate and translate two-dimensional meshes into three-dimensions. There are specialised algorithms for three-

dimensional mesh generation.

Because of the previously mentioned limitations, we must develop a method of extracting the layer boundaries from the mesh so that we can use these contours as building blocks for advanced models which will be re-meshed correctly for their dimensionality and complexity. Our solution to this challenge is the implementation of the Silver Linings module in Section 4.5. Silver Linings satisfies the third primary objective: Create a module which allows the designer to extract FLOOXS contours for three-dimensional model generation.

Supporting geometrical operations are required for us to create shapes that are combinations of multiple contours. We discuss the implementation and description of the required operations in Section 4.6.

We can begin to create three-dimensional circuit models once all the supporting tools are complete. These geometrical operations satisfy the fourth primary objective: Implement tools which allow a designer to combine extracted contours to generate more complicated structures.

We describe the method for manually modelling a superconducting cell in Section 4.8.2. One final useful function of Katana is the capability to calculate the volumes of all physical volume elements in a Gmsh mesh file. We describe the volume calculation algorithm in Section 4.7.

See Fig. 4.1 for a flowchart of 3D model generation using Katana.

Katana's functionality is divided into five key modules:

1. GDSCpp library
2. Cross-section generator
3. Geometry file modelling tools
4. Meshfile operation handler
5. Automatic 3D Cell Modeller

Katana makes use of the first four modules when a designer manually creates a 3D model using the design flow specified in Fig. 4.1.

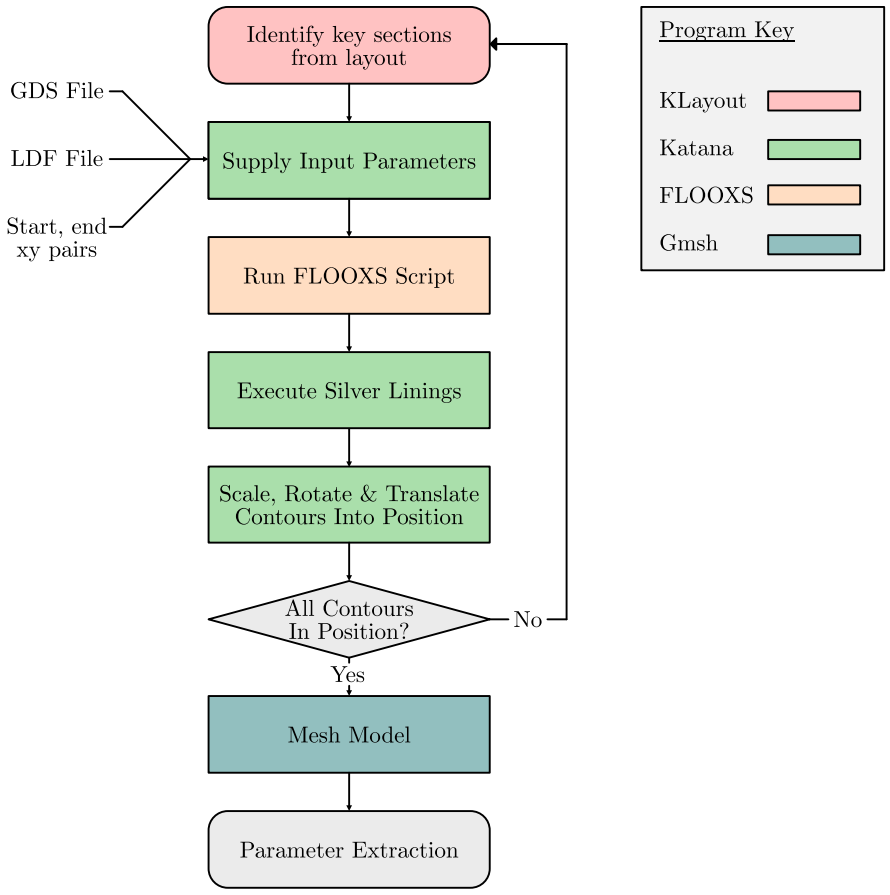


Figure 4.1: Flowchart of 3D circuit modelling.

4.2 GDSCpp

GDS files are binary-encoded files conforming to the architecture described in the Calma GDSII stream format manual [51, 52]. Mask layouts are described hierarchically according to parent structures and child elements. The GDS file is simply a sequential stream of these structures with their elements and other structure references. A structure can hold the following elements:

- Boundary
- Path
- Structure reference
- Array reference
- Text
- Node
- Box

A boundary is a polygon defined by a list of xy coordinate pairs, the layer number, starting coordinates and scale/rotation modifiers. A box is the same as a boundary but may only have five elements with the first being equal to the last. Paths are essentially lines of defined thickness and end cap type (round, flat). Structure references allow designers to repeat the entire contents of a structure and place/rotate/scale it. Array references are essentially a grid comprised of repeated structures. A node can be thought of as a coordinate flag. Text is useful for marking features in the file.

GDSCpp is, at its heart, a translation system. Fig.4.2 reveals the information flow which the library controls. When a programmer creates a GDSCpp object, they generate a blank representation of a GDS file in memory. The import function accepts a path to the GDS file. Each record is decoded from the stream and immediately imported into the object, which holds a C++ vector of structures and other basic properties such as the file name and version number.

The programmer can add, remove or modify the structures and elements contained in the object using the public functions exposed by the class definition. A programmer can also use the library to generate GDS files from scratch using GDSForge's functionality. GDSForge is the sub-module which creates structures and elements. It is especially useful for programmers that need to generate GDS files parametrically. One program which already uses GDSCpp is ViPeR, which places and routes cells to create advanced circuit components. Fig. 4.3 shows the layout of a 4-bit Kogge-Stone adder (KSA), generated by ViPeR [53, 54]. For a sense of scale, an OR gate has been encircled. See Fig. 4.4 for a Klayout GDS view of the OR gate. The gate's length

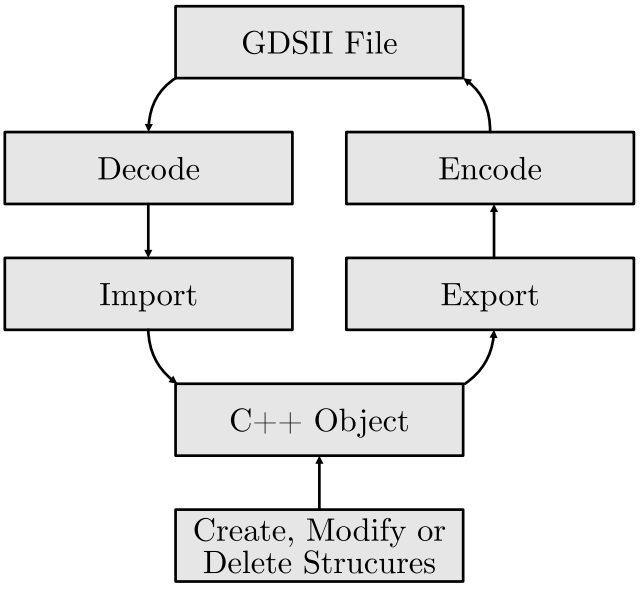


Figure 4.2: GDSCpp library functionality.

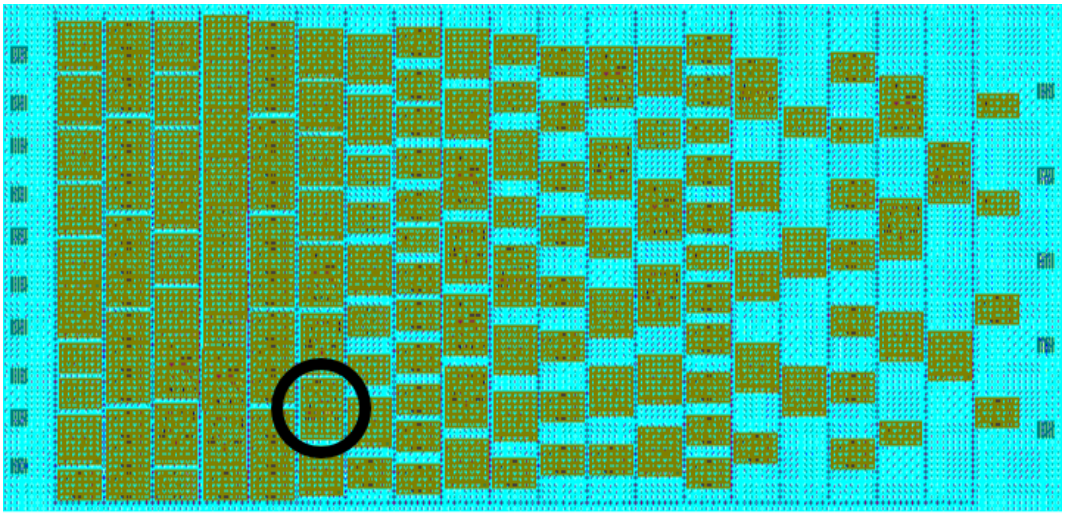


Figure 4.3: Kogge-Stone adder with OR gate encircled.

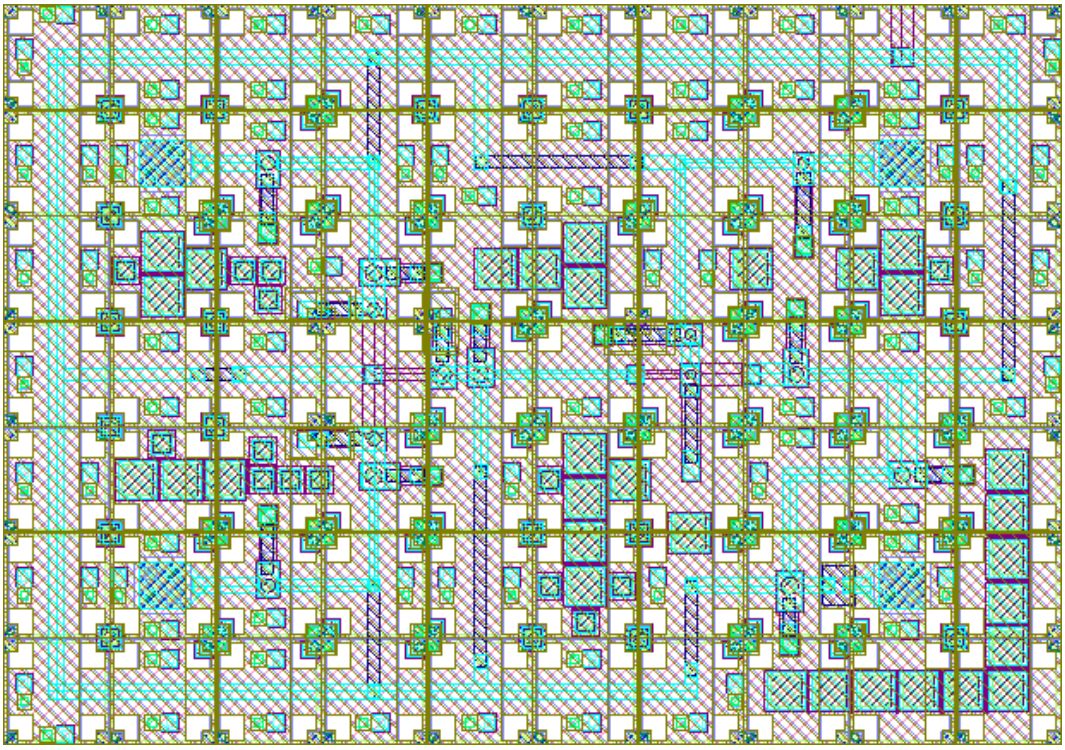


Figure 4.4: OR gate [55].

and breadth are 100 µm and 70 µm respectively.

GDSCpp can also be used to decode and translate the entire binary stream to ASCII text or to generate a hierarchical diagram of all the structures and their dependencies. When we import a file or create a structure, the bounding box is calculated automatically. The bounding box information can be used as a parameter for GDS file analysis. An example of such an analysis could be

filtering polygons according to a set region.

4.3 Cross-section Generation

4.3.1 Overview

The cross-section generator accepts a two-dimensional line's start and end co-ordinates as an instruction to create a window into the circuit. The window must be as tall as the circuit being analysed and as wide as the length of the cross-sectional line. Fig. 4.5 illustrates the process. Fig. 4.6 indicates the program flow from user input to generated files.

The layer definition file (LDF) is a file format developed initially for InductEx which holds fabrication-specific information to augment the mask layout. The LDF specifies information, such as layer thickness, what type of material the film is made of and the sequence of the layer's deposition in the fabrication process. After the section coordinates and GDS/LDF paths have been supplied, Katana imports all of the mask information for each layer of the fabrication process.

4.3.2 Population of Cross-section Data

The cross-section population function tasks are described by Fig. 4.7. The first operation that the cross-section population function executes is determine what type of cross-section is being investigated. The designer-supplied cross-section can be parallel to the x-axis, y-axis or parallel to neither. affecting the straight line equation,

$$y(x) = mx + c\,, \tag{4.1}$$

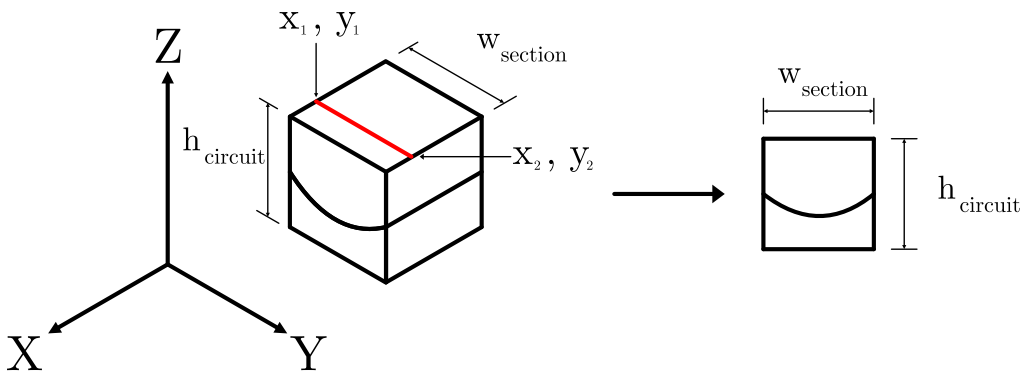


Figure 4.5: Cross-section generation outcome.

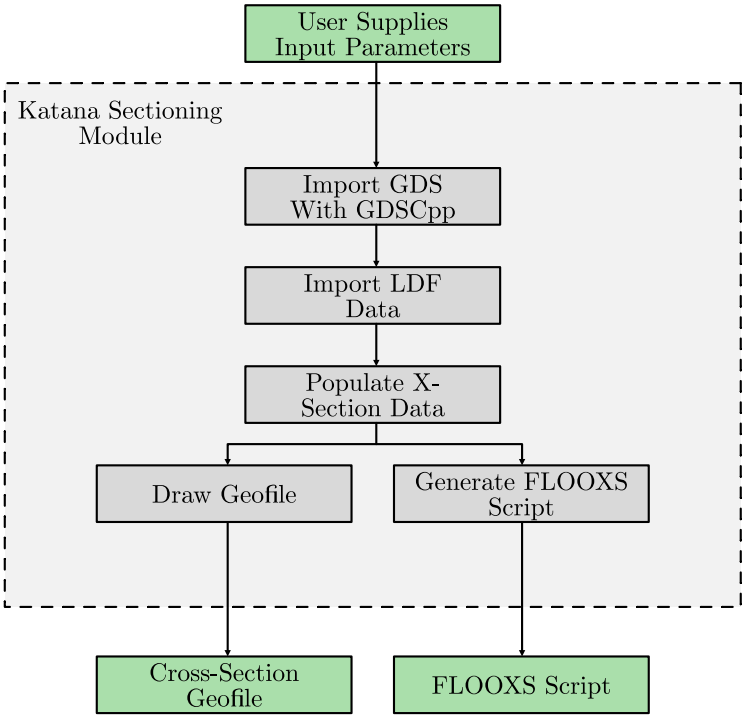


Figure 4.6: Sectioning module design flow.

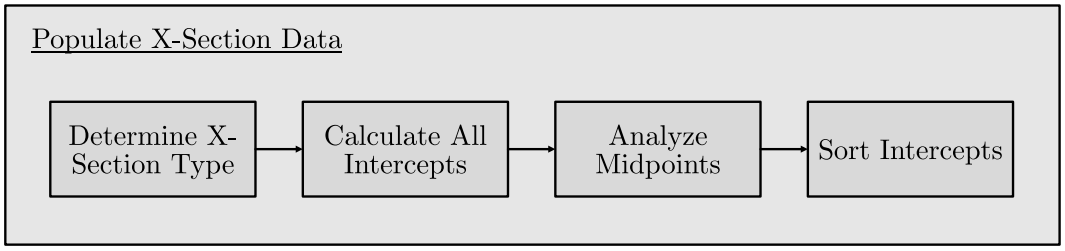


Figure 4.7: Population of cross-section data.

where y is the function output, m is the gradient, x is the function input and c is the value of y when x is set to zero. The gradient is calculated as

$$m = \frac{y_{p2} - y_{p1}}{x_{p2} - x_{p1}}, \tag{4.2}$$

where y_p and x_p are the coordinates of any point on the line. The straight line equation cannot be used for each case required by Katana as division by zero is undefined. Vertical lines have a gradient tending towards infinity and horizontal lines have a gradient of 0. The relevant equation can be selected for the next step by determining the orientation of the straight line, which is either parallel to the horizontal axis, vertical axis or neither.

Katana takes every single GDS structure, including structure and array reference elements, and "unpacks" them into their absolute coordinates. GDS paths are converted into polygons. For each layer, the bounding box of each

polygon is compared against the coordinates of the cross-section line. If the line intercepts or lies within the bounding box, the shape is marked for analysis. All marked polygons are then analyzed.

Each marked polygon is defined by a list of xy coordinate pairs, which are turned into a list of lines. The list is formed by grouping a point in the index with its successor. The last point is paired with the first to conclude the polygon. Each polygon line is classified by its gradient or lack thereof, just like the user-supplied cross-section line. Every line of a marked polygon is then compared to the cross-section line.

Katana first determines the point of intercept of the two lines as if they never terminated in either direction. The program then confirms whether the point of interception lies within both boundaries of the specified lines. There are a total of 9 different potential interactions between two lines:

1. Horizontal-horizontal
2. Horizontal-angled
3. Horizontal-vertical
4. Angled-horizontal
5. Angled-angled
6. Angled-vertical
7. Vertical-horizontal
8. Vertical-angled
9. Vertical-vertical

In the above list, we always order the lines so that the polygon's line is first (left) and the designer-specified cross-section line is second (right). We fully describe the point-of-interception calculation process for a single case, case four, to explain the process. Each case has different boundary conditions and a slightly different method for solving the point of interception, but the principle remains the same.

When the polygon's line is parallel to neither axis, a finite gradient value exists, so

$$y(x) = m_{poly}x + c_{poly}\,. \tag{4.3}$$

The subscript *poly* indicates that the line belongs to the polygon. Because we know the cross-section line is parallel to the x-axis, the gradient factor falls away and the equation for the second line is simply

$$y(x) = c_{xsec}\,. \tag{4.4}$$

The subscript $xsec$ indicates that the line belongs to the cross-section. Two non-parallel lines will always have a single point of intersection which can be determined by equating both straight line equations:

$$m_{poly}x_{intercept} + c_{poly} = c_{xsec}$$

$$x_{intercept} = \frac{c_{xsec} - c_{poly}}{m_{poly}} \,. \tag{4.5}$$

It is also useful to derive the equation for calculating c by using the alternate form of the straight line equation:

$$y - y_1 = m(x - x_1)$$

$$y = mx + (-mx_1 + y_1)$$

$$\therefore \; c = -mx_1 + y_1 \,. \tag{4.6}$$

Any point along the line can be used to substitute for x_1 and y_1. With $x_{intercept}$ solved for, $y_{intercept}$ can be determined by back-substitution into either line equation. In this specific case, it is clear that $y_{intercept}$ must be the y value which defines the horizontal line, as Eq. 4.4 is independent of x.

We now have the point of intercept, but do not know if it lies within the confines of the finite lines, depicted in Fig. 4.8. As a way to simplify the analysis, We order the cross-section line coordinates so that x_{xsec_1} is always less than or equal to x_{xsec_2}. We can now see where the point of intersection must lie in order to be considered valid. For this case, the inequalities of Fig. 4.9 must be satisfied. When the polygons are read in, they are checked for validity. Therefore, there will be no case where p_{poly_n} is ever equal to p_{poly_m} where n and m are indices of any point in the coordinate list. If the intercept point is valid, it is retained, and the distance from the first point of the cross-section to the point of intercept is calculated.

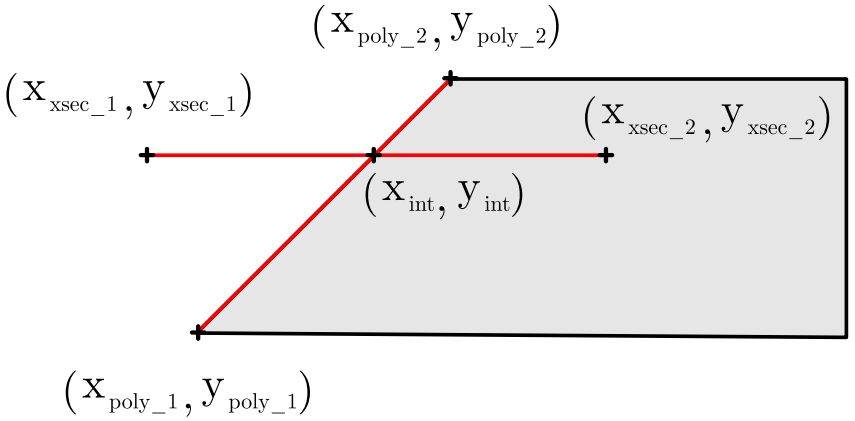


Figure 4.8: Analysis of single polygon line segment against cross-section.

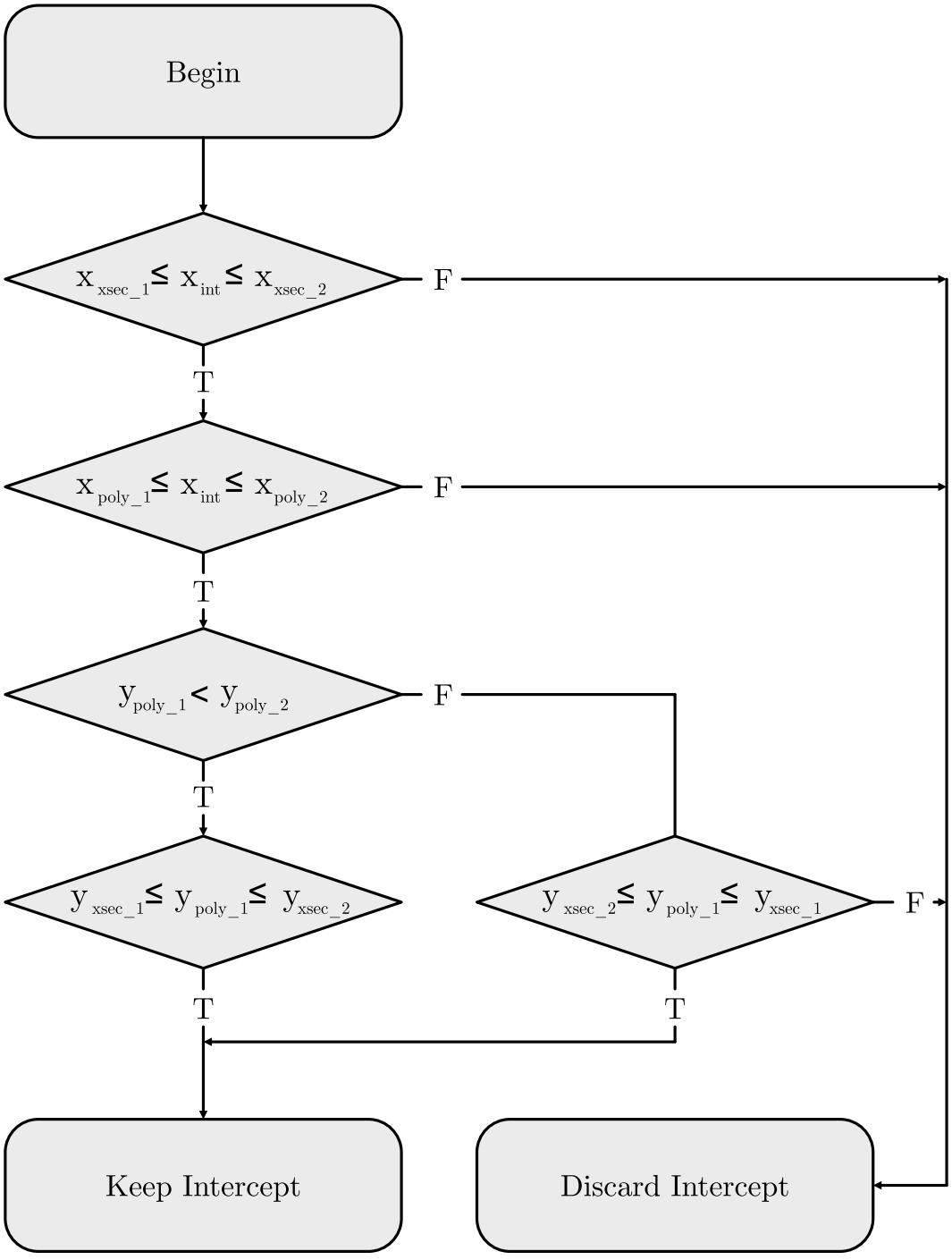


Figure 4.9: Bounding inequalities of angled-horizontal line interaction.

Once all of the points of interception for the layer polygons have been calculated, we sort them in ascending order according to their distance from point $xsec_1$, remove any duplicate entries and add the start and endpoints of the section to the list. At this stage of the process, we know where all the material boundaries are, but we do not know whether the material between two intercepts should be positive or negative. Additionally, positive and negative materials have different meanings depending on the mask type defined in the LDF file. Therefore, we need to individually analyse each segment demarcated by two successive intercepts. This analysis is achieved by utilizing the point-in polygon problem for the midpoint between each intercept. Fig. 4.10 visualizes the material determination process. The symbol P_{int} refers to a calculated point of intercept. The red stars represent segment midpoints. If the midpoint lies within the bounding box of any of the polygons specified for the layer, the point-in polygon problem is executed for that polygon. If the point is found to lie within the polygon, we consider the segment to be positive.

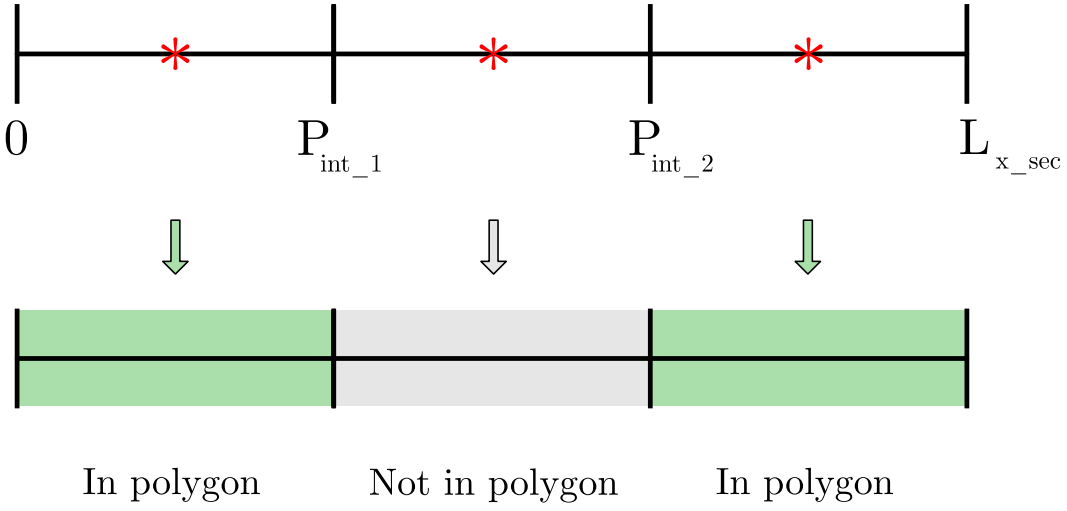


Figure 4.10: Determination of material type for each layer segment.

4.3.3 Point in Polygon Problem

The point in polygon problem can be solved using various methods, but the most robust and generally-used method is an implementation of the winding number algorithm. The algorithm is popular primarily because it does not require a programmer to handle edge cases separately. The winding number, w_n, determines whether a point lies inside of a specified non-simple closed polygon. The algorithm calculates how many times the polygon revolves around the point. We consider a point to be outside of the polygon when there are no revolutions recorded.

The winding number algorithm is easily described by using polar coordinates. A polygon can be defined by piecewise functions when the overall function is piecwise through it's entire domain. With this in mind, we define a parametric equation as a function of time:

$$r = r(t)\,,$$
$$\theta = \theta(t)\,,$$
$$t_0 \leq t \leq t_f\,. \tag{4.7}$$

In order to represent a closed shape, the equation must be continuous and terminate where it started. Therefore,

$$\theta(t_f) = \theta(t_0)\,,$$
$$r(t_f) = r(t_0)\,. \tag{4.8}$$

Because the initial and final positions are the same, $\theta(t_f)$ can only be equal to an integer multiple of 2π. In other words, if we do not fully revolve around the origin, we do not complete our shape. We can now define the winding number, w_n, as

$$w_n = \frac{\theta(t_f) - \theta(t_0)}{2\pi}\,. \tag{4.9}$$

In Fig. 4.11, we consider a continuous curve function $\theta(t)$. If a user follows the path of the function, we see that it revolves clockwise around the origin once between t_0 and t_f. If we plug the θ values for t_0 and t_f into Eq. 4.9, we see that w_n is negative one. Therefore, the origin lies within the polygon as the w_n is nonzero. Fortunately, the principles of the algorithm can be implemented rather easily by considering the following: First, draw any straight line from a starting point which you wish to analyze and extend it through the rest of

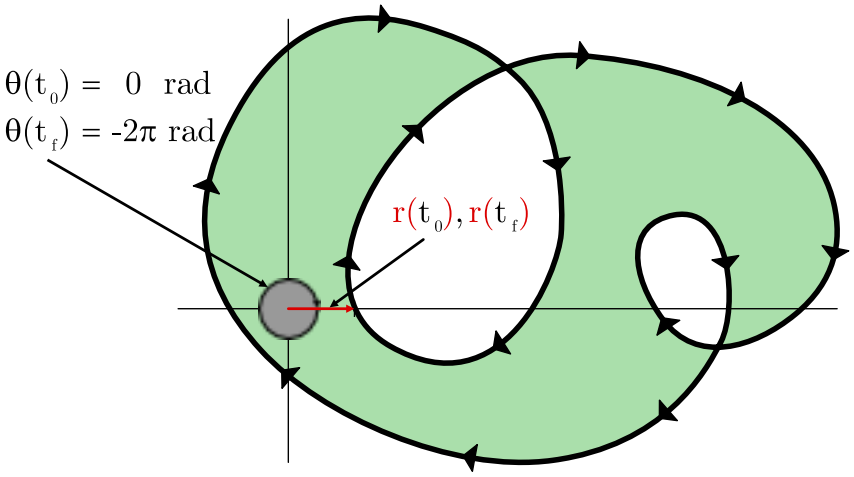


Figure 4.11: Example point in polygon problem.

the polygon. Start counting from zero. This count is the winding number. Every time the polygon crosses the drawn line in a clockwise direction relative to the point, subtract one from the count. Every time the polygon crosses the line in an anticlockwise direction, add one to the count. Once all intercepts are accounted for, if the count is non-zero, the selected point lies within the polygon. The relative orientation of a polygon's line segment about the point in question is determined through the use of a cross product for two-dimensions in a right-hand coordinate system:

$$a \times b = \begin{vmatrix} \mathbf{i} & \mathbf{j} \\ a_x & a_y \\ b_x & b_y \end{vmatrix}$$

$$a \times b = (a_x b_y - a_y b_x)\mathbf{k} \tag{4.10}$$

In the case of the point-in-polygon problem, a represents a vector from the test point towards the starting point of a polygon vertex vector and b represents the polygon vertex vector. If the sign of the resulting cross product is positive, then the polygon's line segment is oriented in a counterclockwise direction about the point in question. If it is negative, then the vertex vector is oriented clockwise.

4.3.4 Drawing the Geometry File

It might prove useful for a designer to generate a cross-section without running fabrication process modelling. We implemented such a method to display the cross-section in Gmsh. The 2D geometry file (.geo) is an ASCII file which adheres the commands described in the Gmsh manual [56]. Figures 4.12 and 4.13 contain a flow diagram of the layer-by-layer cross-section geometry construction process and a visualization of the process respectively. The two-dimensional cross-section consists of points (P), lines (L) and plane surfaces (S) which can be assigned colours and physical properties if required.

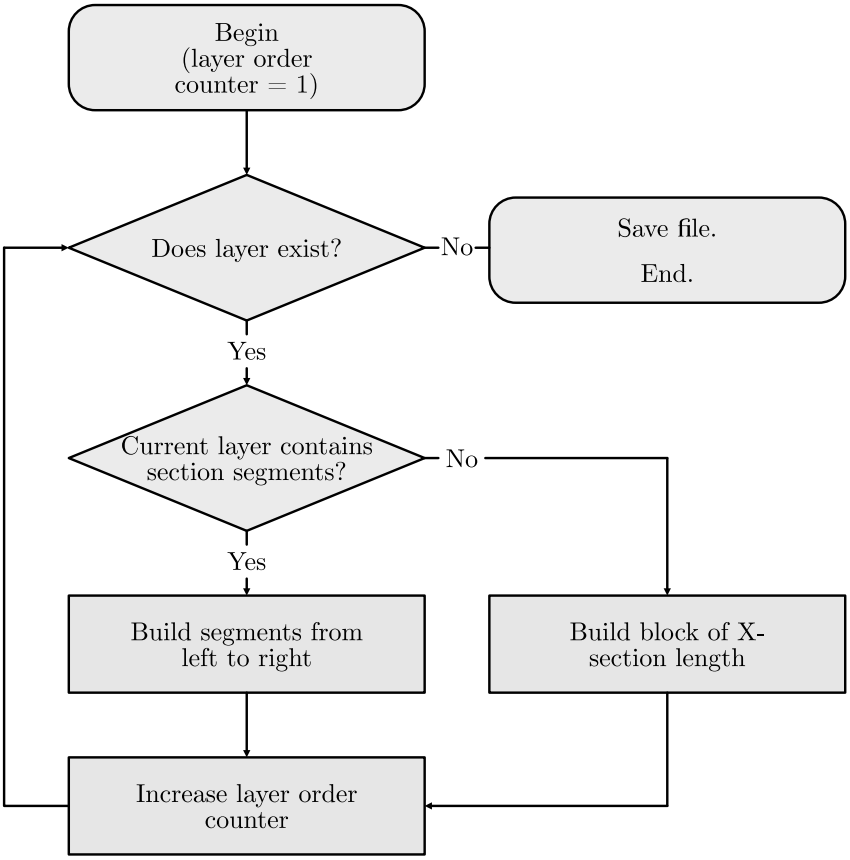


Figure 4.12: Flow diagram for construction of cross-section in geometry file.

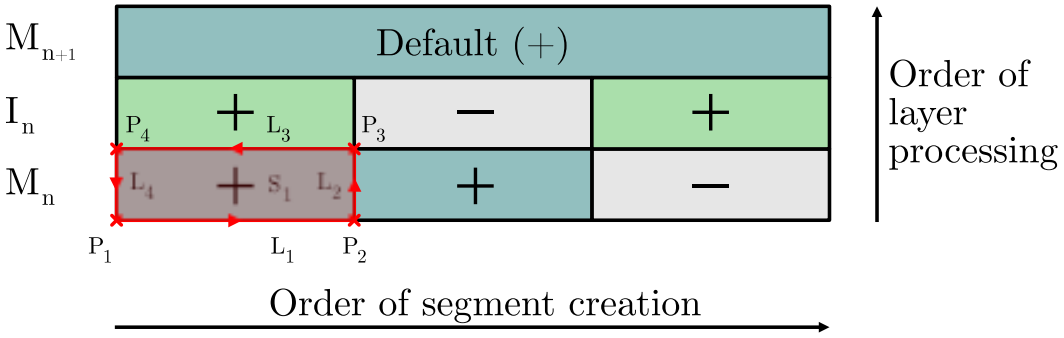


Figure 4.13: Diagram indicating geometry file construction process.

4.3.5 Cross-section Sample

In Appendix B, we show the terminal output and generated code of a cross-section command using the coordinates specified in Fig 4.14. Katana prints, per layer, a list of all boundaries, points of intercepts discovered and the distance of each intercept along the cross-section. The program writes a geometry file (*.geo*) to disk. An abridged version of the geometry file is shown in Appendix Section B.2. A Gmsh rendering of the file, Fig. 4.15, reveals that interconnects are coloured green, metal layers blue and etched areas are gray. In true fabrication, the material of the next layer in the sequence flows into the etched area. Due to the complexity of depicting the fill and planarisation, material flow is not handled in the geometry file. However, it is handled by the Katana-generated FLOOXS script in the next section.

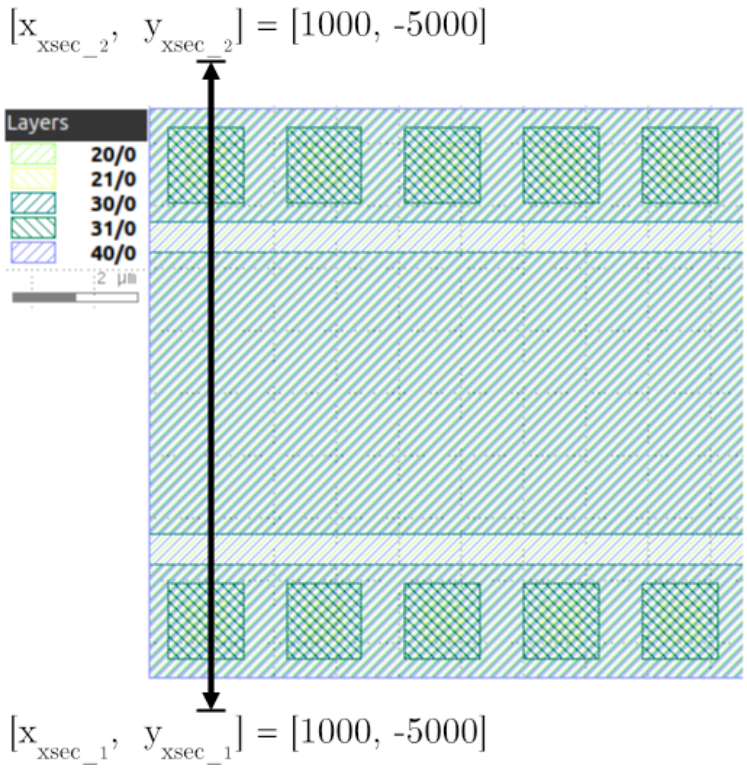


Figure 4.14: Klayout GDS View of PTL.

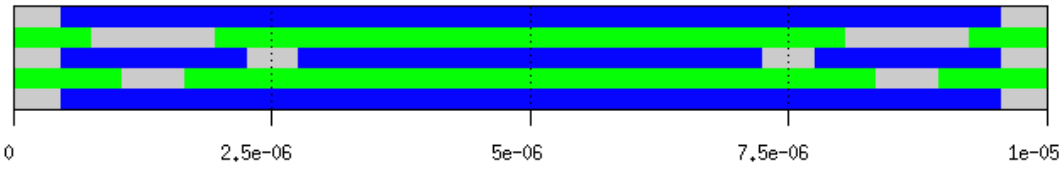


Figure 4.15: Gmsh view of generated cross-section through PTL.

4.4 FLOOXS Script Generation

4.4.1 Process description

Fig. 4.6 revealed that Katana generates two scripts from every cross-section. By using the same cross-section as the one in Fig. 4.14, we generated a FLOOXS script to model the PTL. Katana automatically generates commands for FLOOXS to run. In a sequential order, Katana:

- Configures a default grid space for FLOOXS
- Instructs FLOOXS to use Gmsh as the meshing software
- Initializes the simulation grid
- Creates a base region where deposition can begin from
- Defines where masks must be positioned
- Creates a viewing window of default dimensions
- Instructs FLOOXS with a sequence of etching and deposition processes

The designer can alter the parameters of the script to suit their needs. For instance, if they wished to have a more refined mesh, they could reduce the values of the spacing commands at the top of the script. They could also choose to use the default meshing system of FLOOXS, Triangle [43], by commenting out the *pdbSet* command.

Katana's script makes use of the same system as was previously described in Fig. 4.10. However, instead of depicting the segments as blocks as it does for Gmsh, it defines masks for use in the etching process. These masks are given the same name as the layer in which they occur. The deposition commands control a layer's thickness. The required layer height is obtained from the LDF file. The deposition and etch commands deposit and remove material at a rate of 100 nm/s. We show an example of mask initialisation in the FLOOXS script of Appendix Section B.3.

4.4.2 Stability of FLOOXS

Because FLOOXS is still being adapted from semiconducting simulation towards superconducting purposes, the program is mostly unusable for larger cross-sections and runs into stability errors when simulating multiple layers. Therefore we have had to reduce the complexity of our simulations and combine multiple simulations to model the PTL successfully.

In Fig. 4.16 we demonstrate the successful deposition and etching of one side of the PTL. This is the same edge that was modelled in a video [57], which describes Katana functionality in the context of work presented for ISEC

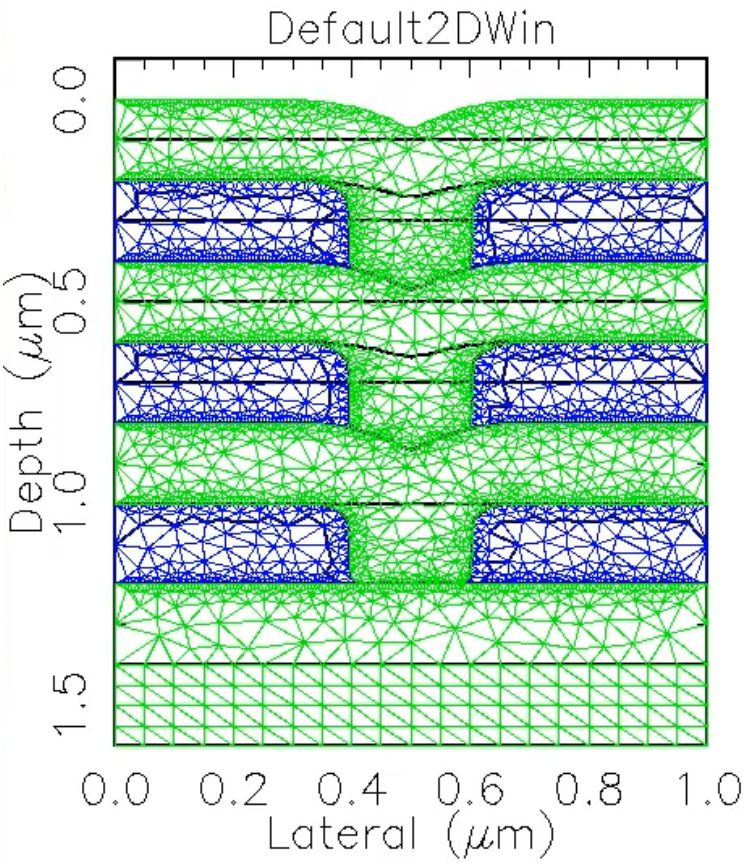


Figure 4.16: FLOOXS PTL edge simulation.

2019 [58]. The curved profile of anisotropic etching is visible on the edges of the metal layers. Silicon dioxide flows into the etched regions successfully.

4.5 Silver Linings Mesh Boundary Extractor

4.5.1 Mesh Generation

Meshing is a process of subdividing a geometric space into smaller elements. The process splits the domain into a discrete number of elements which we can use to determine a solution to a numerical problem. In a two-dimensional plane, the most commonly used mesh elements are usually either triangles or quadrilaterals. In three dimensions tetrahedra and hexahedra are the most common. Triangle, the program which FLOOXS uses utilizes Delaunay triangulations to produce a mesh of triangles for each surface.

4.5.2 FLOOXS Meshing

FLOOXS initially did not have a command for exporting only the contours of each layer the program generated. We, therefore, had to implement a solution to extract the relevant contour information from the mesh which it was capable of exporting. FLOOXS exports the mesh into a Gmsh V2.208 format.

4.5.3 Mesh Format

A Gmsh mesh file has a *.msh* extension. The file can be stored in binary or ASCII format. FLOOXS utilizes the ASCII Option. The format describes a mesh by using the three categories of Fig. 4.17.

Nodes are the lowest-level component of a mesh. Nodes are representations of points in a 3D plane. Each node consists of a positive integer identifier, three axial floating-point coordinates and one optional floating-point number for describing the characteristic length of elements linked to the node. The characteristic length is the maximum size that an element using the node can be.

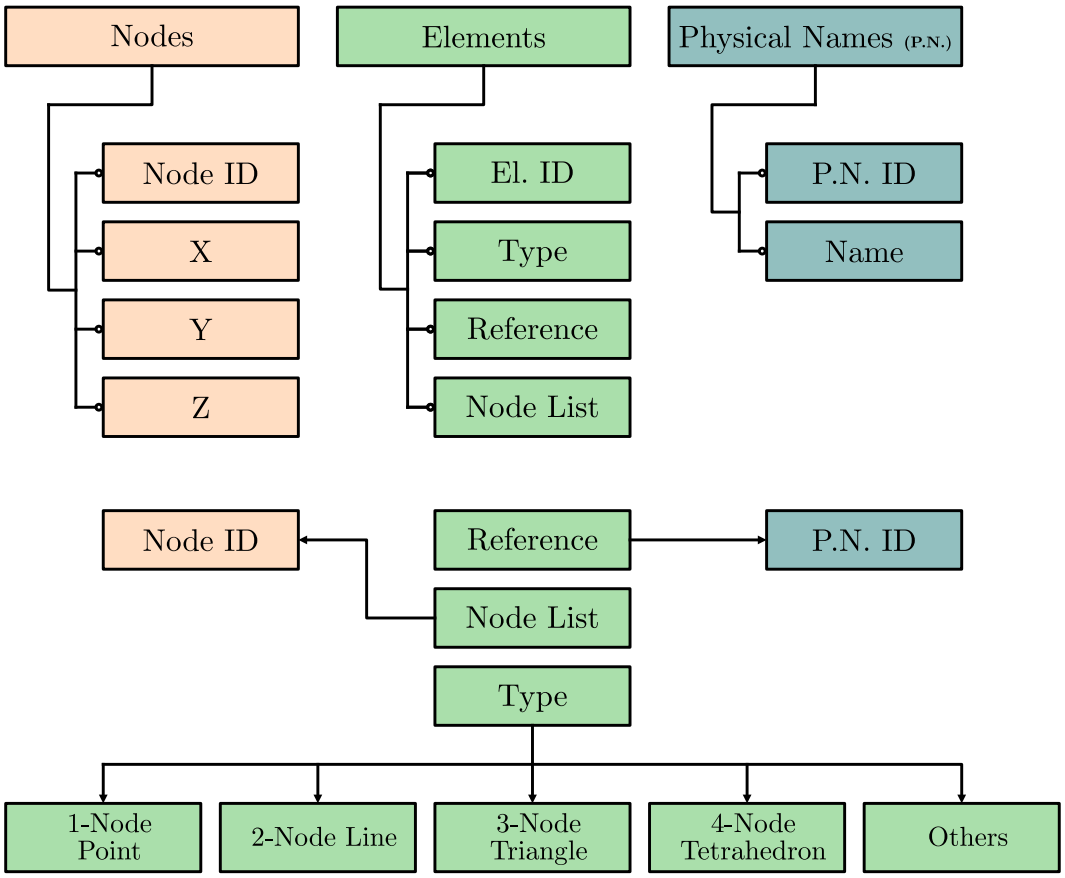


Figure 4.17: Gmsh v2.208 mesh file layout.

Elements are the structures which result from segmenting a geometric region. As previously discussed in Section 4.5.1, for contour extraction, we are primarily interested in 3-node triangle elements. However, Gmsh supports a total of 33 different element types.

Physical names are a method of grouping multiple elements together. They consist of a positive integer identifier, dimensionality integer and a string variable for the name. An element belongs to a physical group, which it references.

We named the Silver Linings (SL) module after the saying, "Every cloud has a silver lining". SL reads in all of the information logged inside of a mesh file and stores it inside of C++ map data structures. We use the identifiers of components as the map keys, allowing for quick, efficient navigation through the dataset. Once we have stored the mesh in memory, we can begin to manipulate the data to extract the boundaries from the file.

4.5.4 The Obstacle of Overlapping Boundaries

FLOOXS starts with a surface defined by a region such as the silicon base in Fig. 4.16, identifiable by its uniform mesh structure. If the designer specifies an etch command, the upper surface of the region is lowered wherever the mask leaves material exposed, reducing the overall area of the etched region.

If the designer specifies deposition, the upper surface is duplicated twice. The second duplicated surface moves upwards with deposition. The second duplicated surface is connected to the first duplicated surface via lines along the left and right boundary of the defined grid. The process of connection creates a new material region. FLOOXS meshes the inside of the new region.

In the case of deposition, because FLOOXS duplicated the upper surface of the previous layer, there is no actual interface between the new and old region. The two regions are considered incoherent. Fig 4.18 depicts the incoherence. We illustrate a physical gap between the first and second mesh. In reality, the surfaces overlap at the interface. We, therefore, need to remove one of the surfaces in order to form a true boundary between the two regions.

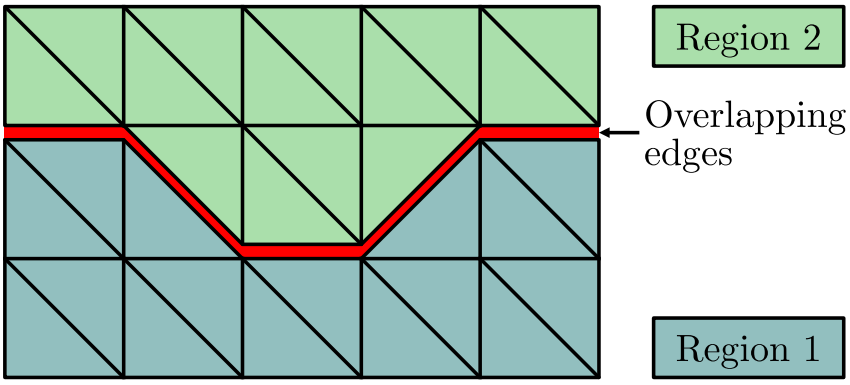


Figure 4.18: FLOOXS mesh of two different regions.

4.5.5 Duplicated Entity Removal

The SL module makes a FLOOXS mesh coherent through the following three-step process:

1. Identify the duplicated node or element
2. Remove duplicated node or element
3. Update dependant constituents to refer to the original node or element

The process is hierarchical. We check for duplicate nodes by comparing the x and y properties against every other node in memory. When an item is found to have the same co-ordinates as a previously defined node, it is removed from the map. The unique identifier of the removed node is stored in a new map of duplicates alongside the identifier of the original. Once duplication identification is complete, we scan through the node lists of all triangle elements and compare them to the duplicates list. If the node list contains a reference to a duplicated node, we replace it with a new reference to the original node. The resulting mesh can now be considered coherent as there is a unique boundary linking the two different material regions to each other.

4.5.6 Edge Classification Methodology

In geometry, we describe a triangle as three vertices connected by lines, called edges. In a Gmsh mesh, a triangle contains three references to nodes which act as the vertices. When we extract the boundaries from a mesh, we only want to retain the edges of all triangles which form the exterior perimeter of the area, which was broken into finite elements. We can only describe an edge by one of the following three descriptors:

- Inner
- Border
- Outward-facing

Internal edges are shared between two triangles which both belong to the same physical group. Border edges are shared between two triangles belonging to different physical groups. Outward-facing edges are only listed once. They are not shared between triangles. A minimum working example (MWE) of a mesh which could undergo boundary identification is represented in Fig. 4.19. $E2$ and $E7$ are inner edges. $E4$ is the sole border edge. $E1$, $E3$, $E5$, $E6$, $E8$, $E9$ are all outward-facing edges.

Silver Linings loops through all triangles, forming edges from their nodes. The program adds each edge to a map of all mesh edges. Each edge in the map consists of a start node, end node and type flag. We sort the edge nodes

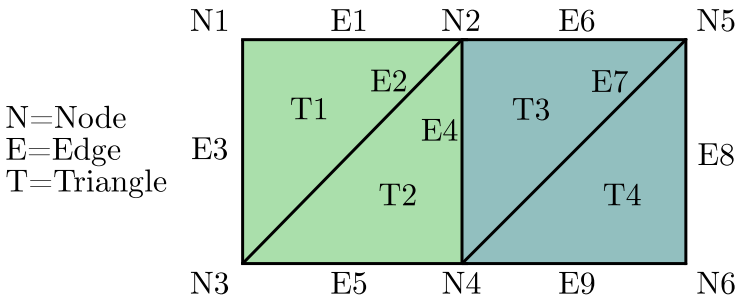


Figure 4.19: Minimum working example of dual-material mesh.

in ascending order. If the edge in question already exists inside the map, the program executes a simple test. It compares the parent material of the stored edge to that of the unstored edge. If the parent material differs, SL sets the stored edge's type flag to represent a border edge. If the parent material is the same, SL sets the flag to represent an inner edge. The default flag is set to represent a border edge, so if it is unchanged by the end of the process, the edge is guaranteed to be a border edge.

4.5.7 Outline Generation

The final step which the Silver Linings module handles is the creation of Gmsh plane surfaces from the available border and outward-facing edges. Gmsh geometry files define a plane surface as a singular reference to a curve loop. A curve loop is a collection of connecting lines, which in turn reference points. If we use a negative sign before the line reference, we swap the starting point with the endpoint of the line, essentially reversing the direction of the line. We must construct a valid curve loop which encapsulates the original material region of the mesh before so that we may define a meshable plane surface.

We construct the curve loop by creating a list of available edges (lines) to use. Each material region may only access the outward-facing and boundary edges of its material type. Each edge may only be used once in the curve loop.

We first select a starting edge, which is the lowest-indexed feasible edge in the list of available edges. Each edge references two nodes. We define the head as the currently-active node in the curve loop creation process. The head begins as the end node of the starting edge.

We compare the head to both nodes of every other available edge, as well as the starting node. Once we have found a matching node, we form a link and move the head to refer to the other node of the linked edge. We set the linked edge to the *used* state and repeat the process, comparing the current head against all other *unused* edges. Once the head matches with the starting node, we know that we have completed one curve loop.

If there are still edges available, there must be another identifiable region. We, therefore, select a new starting node and repeat the process until we have

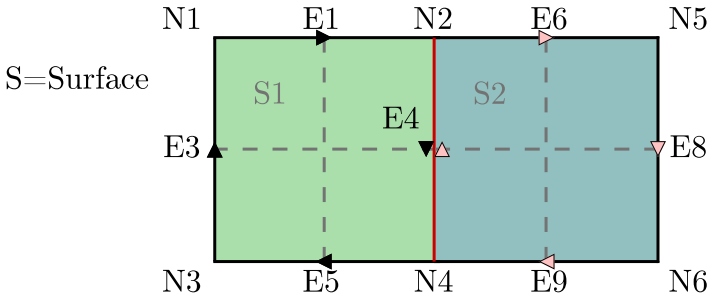


Figure 4.20: Creation of surfaces from a filtered mesh.

identified all curve loops belonging to the current physical group (material type). We run the process for each physical group in the mesh.

We illustrate the process by extending the previous minimum working example into Fig. 4.20. Arrows denote the direction of each curve loop. The first curve loop begins with $N1$. The head is initially $N2$, switches to $N4$, then to $N3$ and then finally matches with the starting node, $N1$, again. Seeing as all of the available edges are used, we move onto the second material and repeat the process. Two curve loops exist in the example. They are used to define $S1$ and $S2$ respectively. For a real example of Silver Linings use, see Appendix A.

In January 2020, we collaborated with UFL and contour extraction was incorporated directly into FLOOXS. The command to export a simulation's contours as a geometry file is

```
struct gmsh_geo_contour_write=output.geo flip.
```

4.6 Geometrical Manipulation

4.6.1 Introduction

We implemented supporting geometrical operations to allow the designer to create three-dimensional models from two-dimensional FLOOXS outputs. The implemented operations form the geometry manipulation module: SMART-C. The acronym stands for scale, merge, append, rotate, translate and make coherent. SMART-C operations are performed on the entire supplied geometry file, as opposed to the built-in operations of Gmsh which only transform selected elements.

4.6.2 Scale

When we scale a geometry file, we multiply every point in the file by a factor supplied by the designer. The operation is simple to implement but proves to be quite necessary when interfacing with FLOOXS because every mesh or contour FLOOXS exports makes use of the program's default units: centimetres.

Therefore, each file must be scaled by a factor of 0.01 to match with systems that store geometry in metres. The designer can also scale the data to match with commonly used database units (nm, µm).

4.6.3 Append

The append function was retired in favour of the merge function when circuit complexity increased. Initially, we wrote the append function to join a second geometry file in the xy plane onto the first. Gmsh does not natively support joining two geometry files together. While there is a merge function, it does not support geometry files that use the same indices. For instance, if the first file has a point with an ID value of 1 and the second file has a point with an ID value of 1, the merge would fail due to conflicting points, even if the points held different coordinates.

The append function shifts the ID indices of the second file by the highest ID recorded in the first file to avoid a conflict. Additionally, all coordinates of points in the second file are shifted along the x axis by the maximum recorded x distance from the origin. In Appendix A, we used the append function to join four PTL FLOOXS-simulated segments.

4.6.4 Coherence

We briefly discussed the coherence between material regions in Section 4.5.5. Functionality initially developed for the Silver Linings module is used in the coherence function. The coherence function, however, removes duplicate entities from all blocks listed in the dependency diagram, Fig. 4.21, as opposed to just nodes and edges.

The coherence function has a cascading effect. By removing duplicate points and redirecting the lines that refer to them, we could, in turn, create

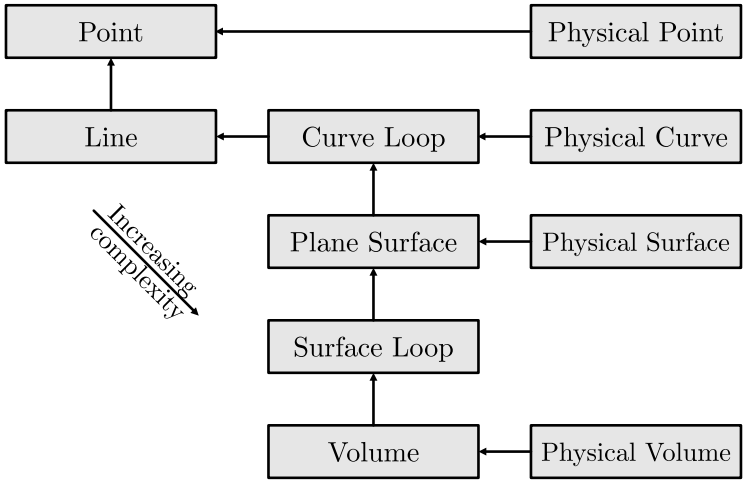


Figure 4.21: Gmsh geometry dependency diagram.

lines which refer to the same points. Therefore, after removing the duplicated points, duplicated lines must also be removed. We repeat the process for curve loops, plane surfaces, surface loops and volumes.

4.6.5 Merge

The merge function is an improved, more versatile method of joining multiple geometry files together. It combines two geometry files regardless of their orientation and dimensionality. The merge function has built-in coherence function calls which ensure that the resulting file has proper interfacing boundaries between plane surfaces and volumes. The function is essentially a combination of elements from the original append function, the coherence function and a simplifying function. Fig. 4.22 contains a flow diagram of the merge function.

The simplify function shifts the ID fields of geometrical indices downwards into the positions of duplicate geometrical features which the coherence func-

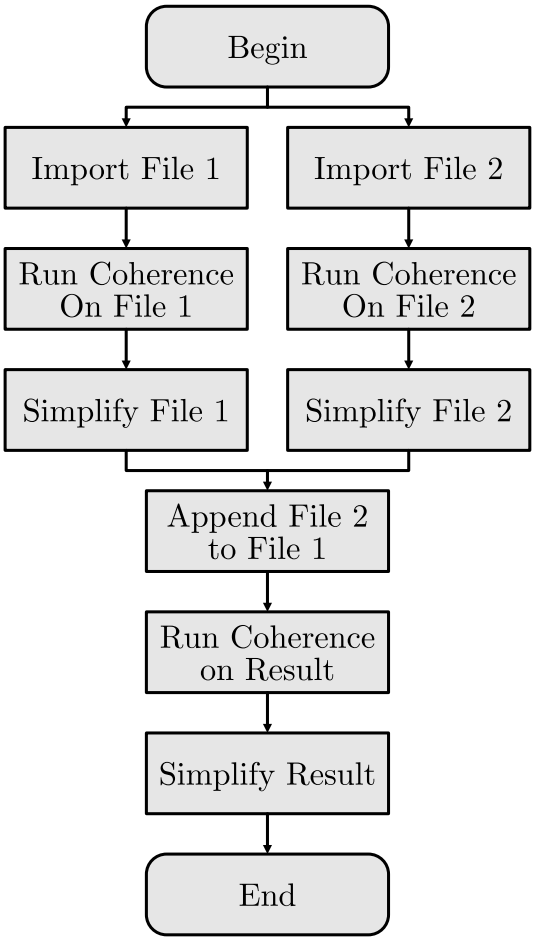


Figure 4.22: Gmsh geometry dependency diagram.

tion deleted. It then updates all references to the old IDs towards the new ones. The resulting file has no index discontinuities.

The append function increases the second file's indices by the maximum value of the first file's indices. We then update all dependency references and join the two datasets together. The combination of the two datasets must also be checked for duplicates to ensure the coherence of the new merged file.

4.6.6 Rotate

The designer specifies the rotation angle, θ, about the desired axis. The rotated coordinates of a point can be calculated using

$$A_r = RA \tag{4.11}$$

where A_r represents the rotated matrix, A is the matrix representation of the point to be rotated,

$$\begin{bmatrix} x_p \\ y_p \\ z_p \end{bmatrix}, \tag{4.12}$$

and R, the rotation matrix. Each point is rotated about the specified axis by a positive angle θ, conforming to the right-hand coordinate system. The rotation matrices [59] follow:

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\theta) & -sin(\theta) \\ 0 & sin(\theta) & cos(\theta) \end{bmatrix}, \tag{4.13}$$

$$R_y(\theta) = \begin{bmatrix} cos(\theta) & 0 & sin(\theta) \\ 0 & 1 & 0 \\ -sin(\theta) & 0 & cos(\theta) \end{bmatrix}, \tag{4.14}$$

$$R_z(\theta) = \begin{bmatrix} cos(\theta) & -sin(\theta) & 0 \\ sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{4.15}$$

If we rotate the file about a point which is not the origin, then the specified rotation point and all points in the geometry file are first translated by a vector which moves the rotation point to the origin. After rotation is complete, all points in the file are translated back by the same amount.

4.6.7 Translate

The translation function adds the designer-specified Δx, Δy and Δz values to every single point in the geometry file.

4.7 Mesh Volume Calculation

If we consider a tetrahedron comprising of position vectors **a**, **b**, **c**, **d** with respect to the origin, the equation for calculation of the tetrahedron's volume [60] is

$$V = \frac{\left|(\mathbf{a}-\mathbf{d})\cdot((\mathbf{b}-\mathbf{d})\times(\mathbf{c}-\mathbf{d}))\right|}{6}. \tag{4.16}$$

The volume calculation function identifies all the physical names in the mesh file and then calculates the volume of every single tetrahedron in the file. Tetrahedrons are mesh elements and therefore each refer to a physical name. The relationship between an element and a physical name was revealed by Fig. 4.17. We add every tetrahedron's calculated volume to the total volume variable of the physical name that it references. Katana only currently supports tetrahedral mesh volume calculation.

4.8 Manual Modelling Technique

4.8.1 Introduction

The initial 3D models which we created with Katana have all been handcrafted using strategic cross-section slices from Katana. The process heavily relies on Silver Linings to extract contours and Katana's merge function to combine the modelled features. We make use of the process described by Fig. 4.1 to build up simple models, starting with a Josephson junction. Next, we model a ground plane-connected JJ. These preceding models build-up towards the Josephson transmission line model. Gmsh reflection commands reduce the workload as many features were symmetrical about an axis.

4.8.2 Modelling the Josephson Junction

We choose an early model of a JTL and focus on an area where moats surround a JJ. The aim is to create a model which can be used to see how process modelling affects flux trapping analysis. A Klayout GDS mask view of the modelled area can be seen in Fig. 4.23.

Our first 3D model is of a junction with a shunt resistor, created without any process modelling. Fig. 3.12 shows the first iteration of a three-dimensional JJ. This junction is used as a reference to confirm the geometry of the subsequent process modelled JJ. In Fig. 4.24, we show an artistic depiction of the cross-sectional view through the length of the JJ. We decompose the junction into even smaller segments, starting with the interconnect on layer I4.

The profile of the interconnect has been extracted from the FLOOXS model, Fig. 4.25a, and rotated about the edges of the 500 nm × 500 nm block to form Fig. 4.25b.

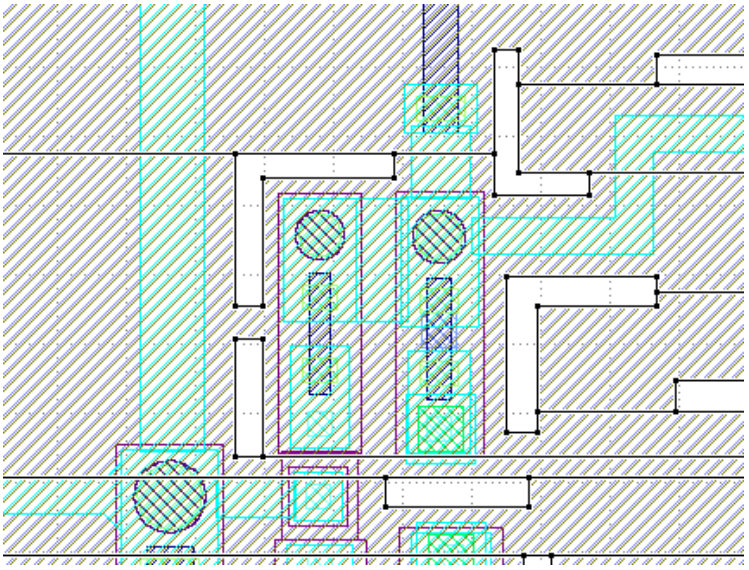


Figure 4.23: Klayout view of modelled JTL area.

In a similar manner, we model the metal segments, which are separated from one another through the etching process. The edge profile of the M5 base electrode is depicted in Fig. 4.26a. At the corners of the segments, we sweep the profile 90° as can be seen in Fig. 4.26. Once the junction is modelled, we begin on the ground plane. The absolute GDS coordinates of the moats are translated into coordinates relative to the JJ. In other words, the origin of the model is the center point of the Josephson junction. The contour is swept around corners in the same manner as is shown in Fig. 4.26, however, the base of the contour is used instead of it's highest point. Additionally, the contour faces outwards instead of inwards. We must also compensate for the position of the material. FLOOXS reveals that an 800 nm wide etch through 200 nm niobium leaves 600 nm between the base of contours on opposite sides. Therefore, if we sweep the contour around the shape, it must be 200 nm narrower

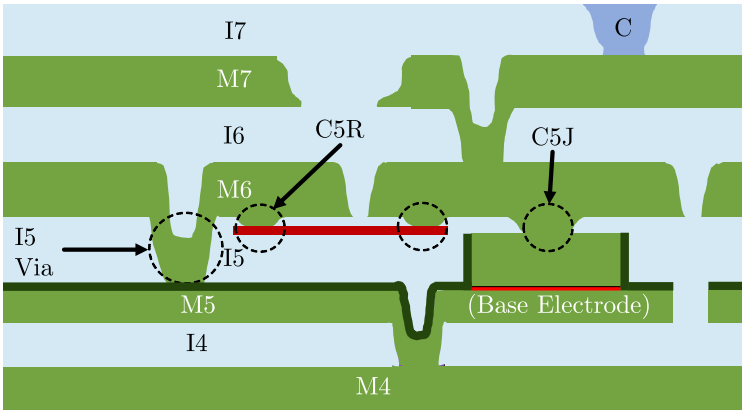
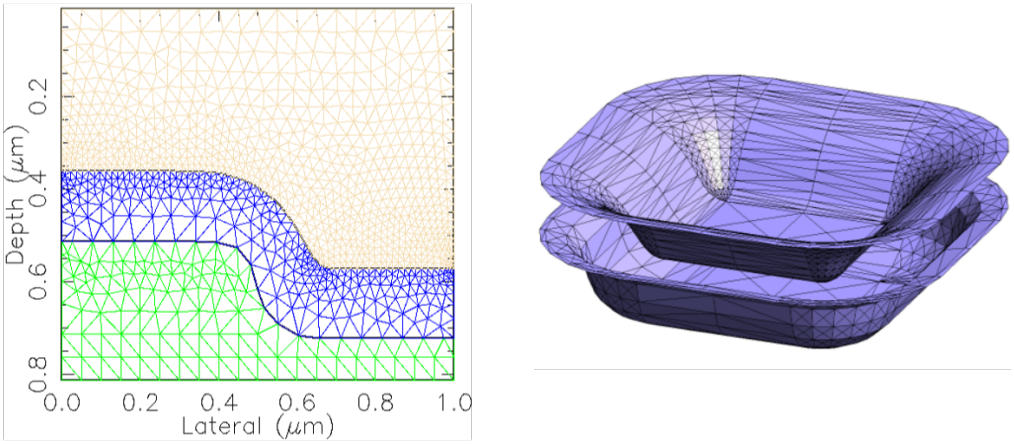
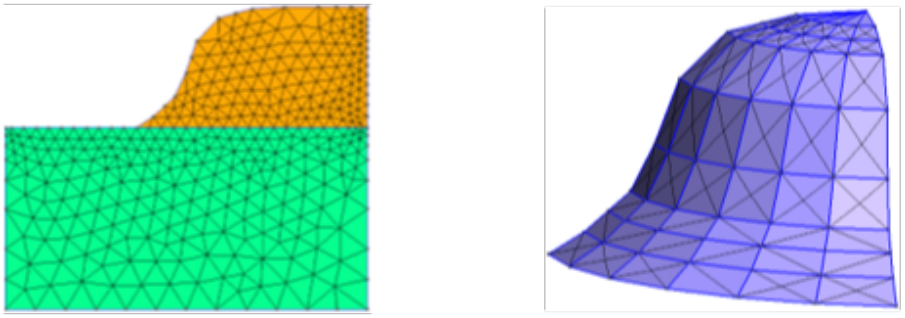


Figure 4.24: Cross-section through the length of a Josephson junction.



(a) FLOOXS interconnect half simulation. (b) Profile after geometric manipulation.

Figure 4.25: I4 interconnect model.



(a) FLOOXS edge profile. (b) Corner profile from edge sweep.

Figure 4.26: Model of M5 metal edge.

than the GDS representation to be true to the cross-sectional FLOOXS profile. An isolated mesh of the top left moat of Fig. 4.23 can be seen in Fig. 4.27

The complete JJ model with its ground plane is presented in Chapter 6, Section 6.3 A full JTL model has also been created with the manual modelling process described in this chapter. The JTL is shown in Section 6.4. The manual modelling process is time-consuming. There are significant segments which can be automated if certain assumptions are made. These observations lead to Chapter 5 where we implement automatic modelling.

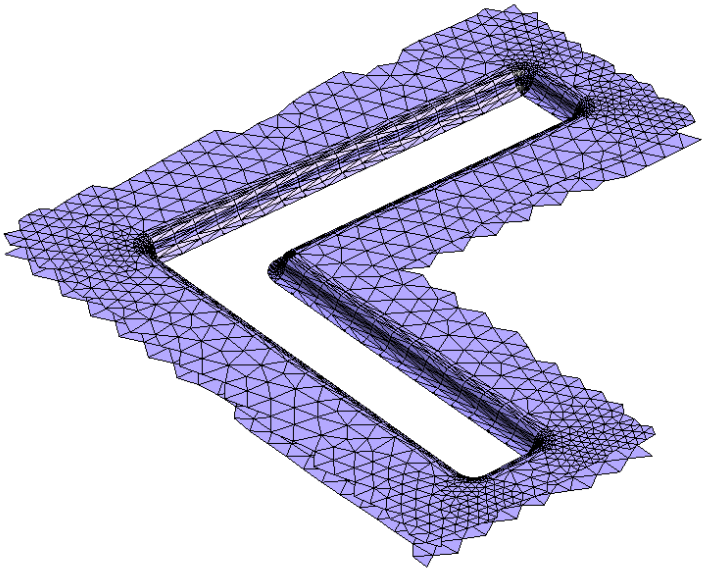


Figure 4.27: Isolated mesh of moat in M4 ground plane.

Chapter 5

Implementation: Automated 3D Modelling

5.1 Automatic Modelling Overview

This chapter describes the implementation of Katana's automatic modelling module. The module creates a meshable, multi-layered, process-modelled representation of a superconducting circuit layout, taking into consideration the fabrication process description. The modeller is capable of handling layouts ranging from a single circuit element to an entire logic gate. Hypothetically, the program can handle even larger layouts. So far, we have only tested it at the gate level to satisfy the project requirements.

Section 5.2 examines the required assumptions which allow Katana to create the three-dimensional model. In Section 5.3, we present the file which describes the fabrication process and by extension, how Katana should represent layers. We then describe the method of creating a blueprint for three-dimensional layer construction in Section 5.4.

Section 5.5 explains how Katana can leverage the FreeCAD API, Boolean operations and constructive solid geometry principles to create a 3D circuit model. We exhibit the method of generating the required shapes for each layer in Section 5.6.

Once the preceding sections have covered the foundational concepts, Section 5.7 explains the design flow, from importing the circuit layout to extracting parameters from the created model. We follow up the explanation with an overview of the Katana-generated Python script, which creates the meshable model. The last two sections discuss the optimizations and shape manipulation required to build fast, functional models.

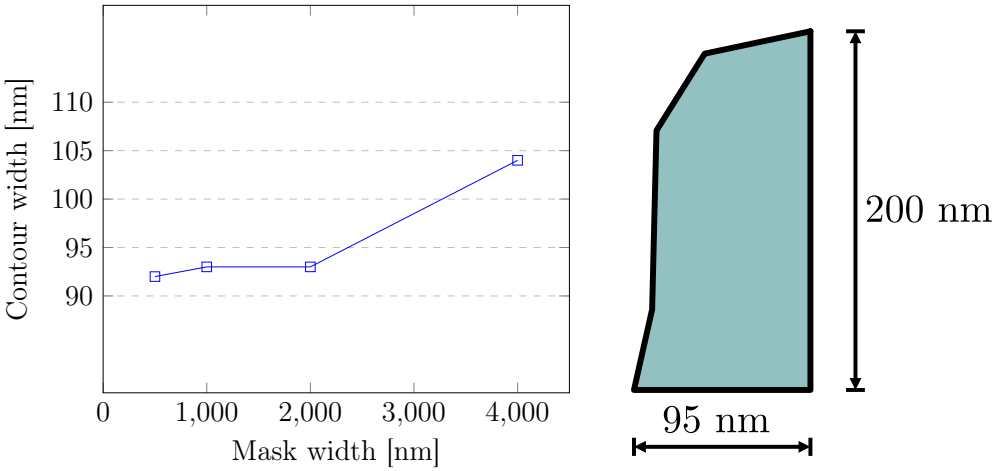
5.2 Modelling Assumptions

FLOOXS simulations of the etching process reveal that the width of an etch contour varies only slightly depending on the width of a mask opening. We have additionally found that FLOOXS forms the same etch profile, regardless of the material type. Silicon dioxide and niobium display the same contour profile for matching mask widths and layer thicknesses. Therefore, we extract a single contour and apply it to all layers of a set thickness.

In Fig. 5.1a, we show the resulting contour widths of various mask openings. Due to design rules, we don't generally design for mask openings smaller than 500 nm. FLOOXS runs into stability issues when simulating etches larger than 4000 nm and design rule density requirements should ensure that we do not generally exceed that opening size. Therefore, we selected the aforementioned values as boundaries in our test scenario. In the case of 200 nm thick layers, we chose the profile seen in Fig. 5.1b, as it sits roughly in the middle of the lower and upper contour width boundaries and should therefore provide the most accurate model if used over the entire layer.

We generated contours for all required layer thicknesses of the fabrication process. After receiving feedback from one of the reviewers of our journal paper, Appendix A, we increased the etch duration used in the simulation to match what is considered standard in the industry more closely. The resulting adjusted profiles more accurately reflect the SEM images [10] of the SFQ5ee process.

Because SEM images can only reveal a two-dimensional view of an etched segment, we are unsure of exactly how corners look. Therefore, we have adopted an approach to corner creation based on how we believe the corners look. However, should future research findings suggest that the corners look



(a) Etch contour width versus mask width.

(b) Selected etch profile.

Figure 5.1: Mask-contour relationship for 200 nm layers.

different, Katana allows for sharper or rounder corner creation. An effective means to create a 3D element is to sweep the selected layer's etch profile about the mask profile of the GDS file. One can intuitively rotate the contour around a convex edge. However, concave edges prove to be an obstacle. Fig. 5.2 shows three potential convex edge types. We cannot use the convex edge node to perform a sweep. If we handle each side of a polygon individually, there will exist an overlapping region (left-hand side of figure) which would cause meshing to fail. In the case of Boolean addition of the two side profiles, this solution would form a mitre edge with a sharp join line which would not make sense in the context of an exposed area surface removal process such as ion bombardment.

If we terminate the edges early and connect them, the 90° corner would be converted into an externally bevelled edge, adding material to the shape as shown in the central configuration of Fig. 5.2. This kind of edge looks more like an approximation than an actual representation of a concave edge. The FLOOXS representations of the etching process would appear to suggest that the configuration on the right-hand side is most likely to represent the real-life scenario of a three-dimensional etch.

FLOOXS simulations show that, even in the anisotropic etching process, a small amount of under-etching occurs. Under-etching essentially translates to an overlapping region between where a mask is specified and where the etch profile begins. Fig. 5.3 visualises the overlapping region. The overlapping region, O_{lap}, is defined as the difference between the specified mask width, M_{dist}, and observed width, C_{start}, where the contour starts. Therefore, to describe a 3D shape accurately, we must compensate for the under-etch by reducing each mask polygon by an experimentally determined value of roughly 20 % of the contour width. Taking all the previous factors into account, we assume that a circuit element will follow the corner conventions displayed in Fig 5.4. The adjusted path is 20 % smaller than the original mask layout, but the contour adds back volume to the circuit element.

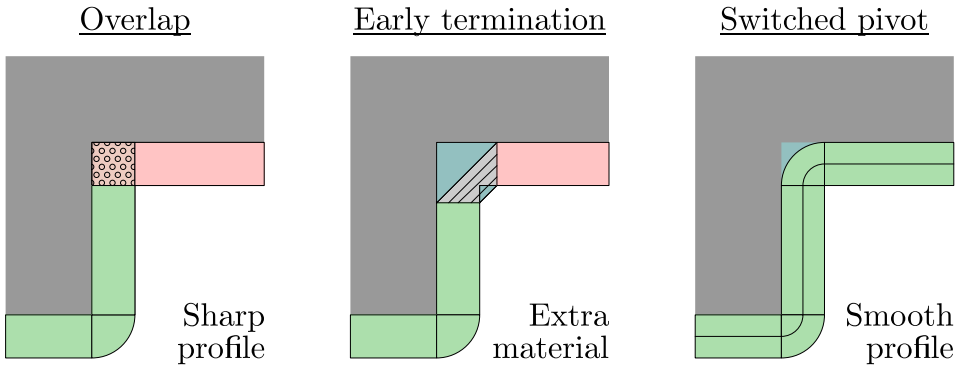


Figure 5.2: Potential convex edge representations.

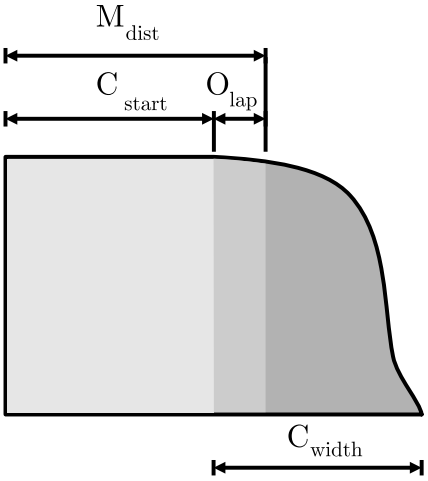


Figure 5.3: Overlap between mask edge and contour start.

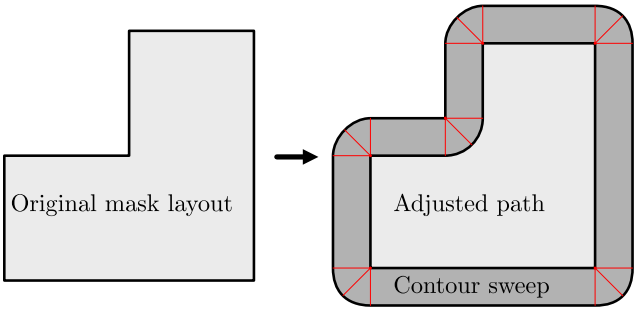


Figure 5.4: Convention for creating 3D circuit elements from 2D contour.

5.3 Process Information File

The process information file, or PIF (*.pf*), is an ASCII information file similar to the InductEx layer definition file utilized in the cross-section generation module. The main difference between the LDF and PIF is that materials are described by their vertical axis base position and layer thickness as opposed to fabrication sequence and layer thickness. This difference in the PIF allows for GDS layers to occupy overlapping regions in the vertical dimension. Layers which overlap include $I5$, $C5R$ and $C5J$ of Fig. 4.24. Listing 5.1 shows an example of PIF entry for the metal layer $M0$.

```
$Layer
Number      =   1
Name        =   M0
MaskType    =   2
ZStart      =   0
Thickness   =   200
Material    =   Nb
$End
```

Listing 5.1: Example Process Information File Entry

The *Number* field corresponds to the GDS file layer number. The *Name* field corresponds to the GDS file layer name. There are four options for the *MaskType* field:

0. Standard.
1. Inverted.
2. Filled.
3. Junction.
4. Auxiliary.

The *standard* configuration means that wherever a polygon is shown in the GDS data for the layer, the layer's material should remain. Therefore, wherever there is no GDS polygon present, the material should be etched away. The *inverted* configuration means that Katana should interpret the polygon data to represent material which should be removed. This configuration is used for isolation layers as niobium through-silicon vias form the minority of the material for the layer. The *filled* configuration indicates to Katana that it should extract the regions where the material is not shown, and convert said regions into polygons which will be cut out of a layer block of the specified material. In the next section, the process followed for Fig. 5.5 is an example of a filled layer: $M0$. The *junction* configuration is specifically designed for the SFQ5ee process. It instructs Katana to place a 9 nm Al-AlOx weak barrier beneath the specified layer. Katana does not generate three-dimensional elements for layers specified as *auxiliary*. The *Material* field specifies the type of metal or insulator used for the layer. *ZStart* specifies the height at which the layer should be generated, and *thickness* specifies the height of the layer and by extension which contour to select.

5.4 Layer Blueprint Creation

Our shape generation algorithm requires a correctly-processed, two-dimensional set of instructions which incorporate the modelling assumptions described in Section 5.2. We call the instructions blueprints because they serve as the plans for how the algorithm will construct the layer in three-dimensions.

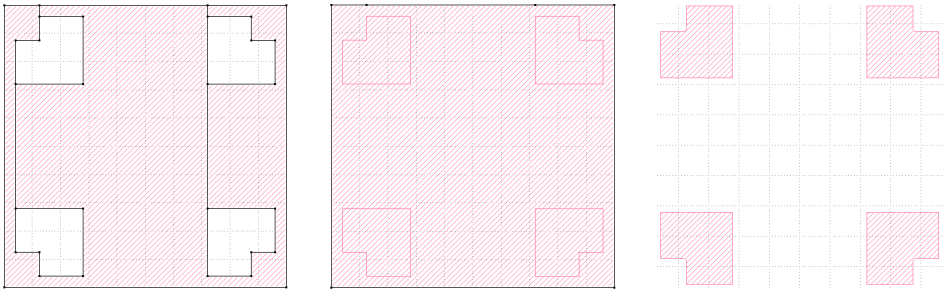
The GDS-compatible convention for representing holes inside of a polygon is to use a continuous line which overlaps but does not cross over itself. Fig. 5.5a shows such an overlapping polygon. The entire shape is represented by one continuous sequence of line segments.

We need to be able to analyse the internal holes as entities separate from the shape they lie within, because oftentimes the shape which contains them is only a portion of the layer's fill structure. The goal is to remove boundaries

which do not contribute to a layer's geometry, such as the boundaries where repeated fill patterns join up with one another. Because of this requirement, we filter each polygon using the Clipper [46] library's *simplify polygon* function. The function removes self-intersections from a supplied polygon, creating a simple polygon; a polygon which does not intersect itself and has no holes. In the case of simplifying a polygon with holes, the function produces new simple polygons for all of the holes and a simple polygon for the surrounding shape. Fig. 5.5b shows the result of polygon simplification on a section of circuit fill structure. Because the polygons no longer follow the GDS convention, the holes appear filled.

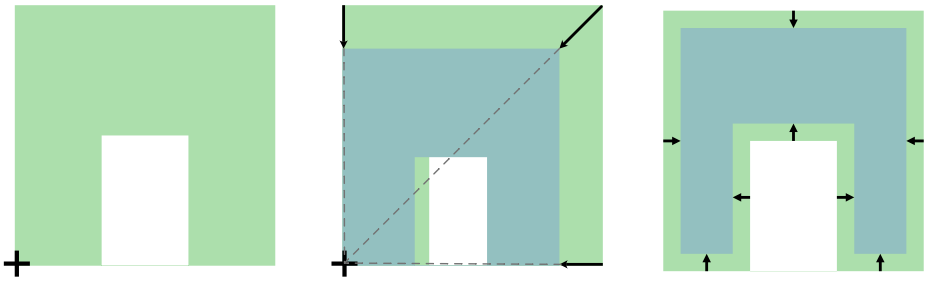
We can model fill layers of a circuit by using a geometrical Boolean cut function which removes a target shape or shapes from the principal one. In the case of the fill, the newly generated hole polygons act as the target shapes. However, after simplification, we have no way of distinguishing which polygons are hole polygons and which are the outline of the original shape. Fortunately, the simplify polygon ensures no polygons overlap. Therefore, it is safe to conclude that if a polygon exists inside of another polygon, it is a hole, unless it is simultaneously inside of two different polygons. The observation forms a repeating pattern which allows us to generalize the statement to: If a simple polygon falls within an even number of other simple polygons, it is a hole. In computer graphics, this observation is commonly referred to as the even-odd rule [61]. We filter out the holes by taking a test point from every polygon and running the point-in-polygon problem against all other polygons of the resulting polygon list.

We now have a list of all polygons which will form the target of the Boolean operation. We must adjust the hole polygons by the overlap factor described in Section 5.2 before rounding their concave edges. Clipper includes a function, *clipper offset*, for expanding and contracting a polygon by a specified width. The function is different from a polygon scaling function. The former takes into account the directionality of each line segment, whereas the latter method is directional about a single point. The differences are shown in Fig. 5.6. For our purposes, *clipper offset* is the only viable choice of the two methods as



(a) Before simplification. (b) After simplification. (c) After filtering.

Figure 5.5: GDS layer simplification process.



(a) Original. (b) Scaled. (c) Clipper-offset.

Figure 5.6: Comparison between polygon size manipulation methods.

every edge must be offset equally by the overlap factor.

Our method of rounding the concave corners is to make use of the *clipper offset's* join type flag. We first expand the shape by an offset equal to the width of the layer's contour and then we set the join type to make use of rounded edges and contract the shape by the same offset. The resulting shapes all have rounded concave corners. We filter away all of the non-hole polygons and are left with the rounded and correctly scaled edges displayed in Fig. 5.5c.

We generate the primary shape for the Boolean cut operation by determining the bounding box of the entire GDS layer and expanding it by five times the layer's contour width in each direction. The expansion ensures that of the model features of the gate are represented completely.

Initially, we automatically constructed the 3D shapes in the Gmsh geometry file format using the fundamental entities described by the dependency diagram of the previous chapter, Fig. 4.21. Each polygon was used as the base for a, (now deprecated), sweeping algorithm we wrote for Katana. The algorithm used each polygon as the base of the 3D shape. Contours were positioned over each polygon point using the convention shown in Fig. 5.4. At the 90° degree corners, we used six contours spaced 15° apart from each other to model the curve. We placed single contours over all the other points of the polygon. Lines and plane surfaces were formed through the identification of repeating number patterns and specific cases for corners, starting and ending points. A polyhedron which was automatically generated by this algorithm is shown in Fig. 5.7. Fig 5.8 zooms in on the corners. The reasons for deprecating this method of three-dimensional shape generation are discussed in the concluding chapter, Section 7.2.

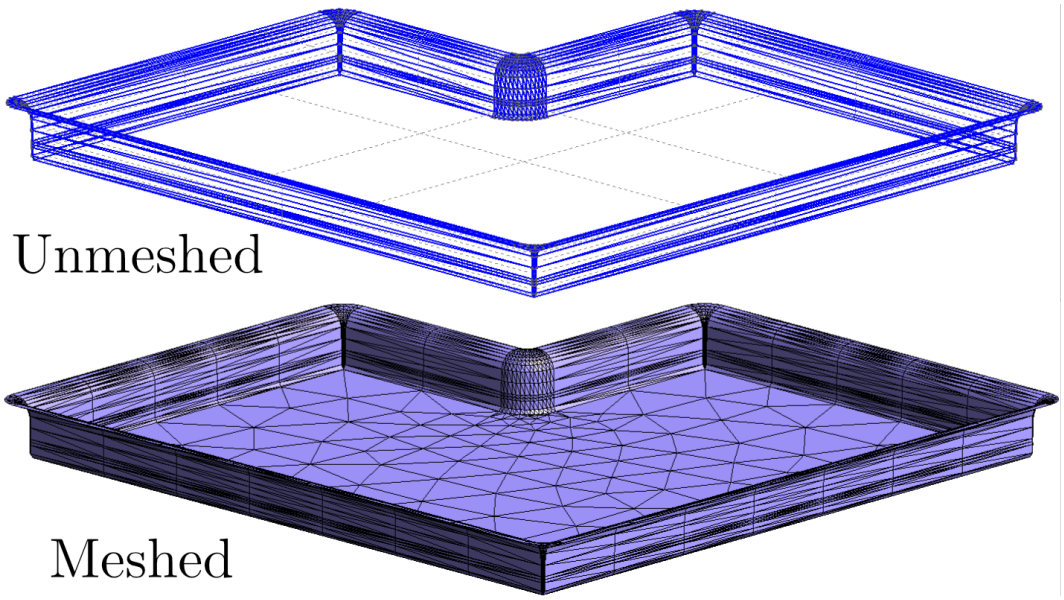


Figure 5.7: Automatically generated hole polyhedron.

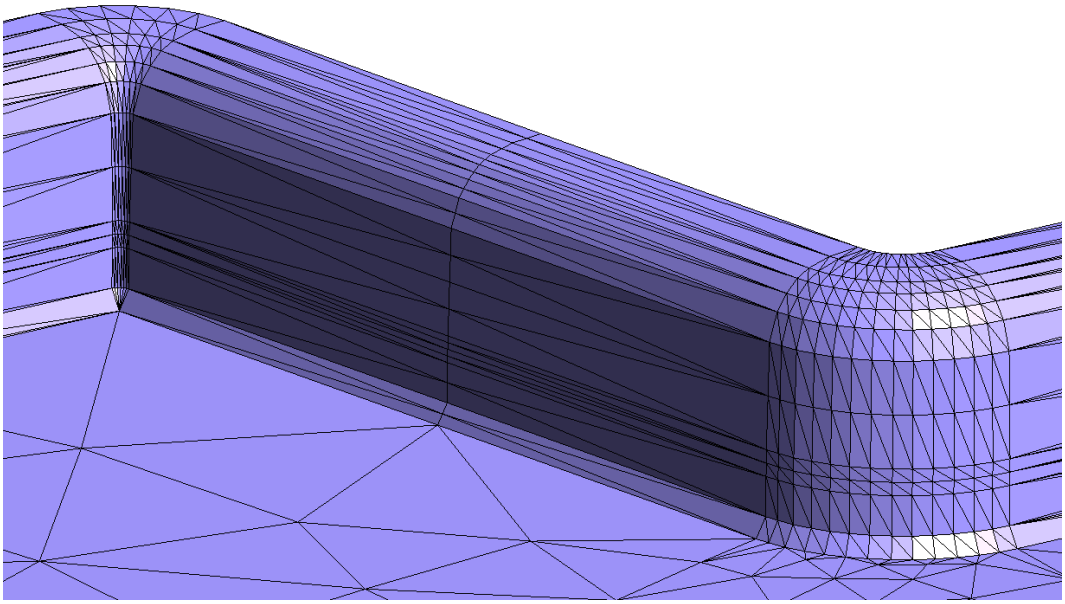


Figure 5.8: Close-up of polyhedron corners.

5.5 Scripting With FreeCAD

The entire backend of FreeCAD is coded as a Python library with some Python-wrapped C++ components. The API is well-documented and highly versatile. The program comes with a comprehensive GUI built with the Qt engine. The GUI converts the functionality of all the buttons to API instructions that feed into the backend, meaning that the functionality of FreeCAD is interface-independent. All actions taken in the GUI can be recorded into a macro; a singular instruction that expands into a collection of instructions to execute a particular task.

The process of building a CAD model of any complexity using constructive solid geometry can be broken down into a set of steps which convert efficiently into API instructions.

FreeCAD can store physical objects in two useful ways, as document objects or as temporary objects in memory. Document objects act as snapshots in time which the designer can interact with inside of the user interface. The document object system forms a tree of parts, operations and their dependencies. These are the objects that are saved in the FreeCAD file when a user executes the save instruction. Document objects are simply objects in memory which have been saved to disk. Fig. 5.9 shows an example tree view of document objects.

Individual layers can be created through a combination of OpenCascade Boolean operations such as *union*, *cut* and *difference*. We can save the layer to the document once we have fully constructed it and repeat the process for each layer. The challenge is, then, to find a way to convert the essence of the algorithm created in the previous section into a constructive solid geometry representation that still makes use of the automatic generation blueprint.

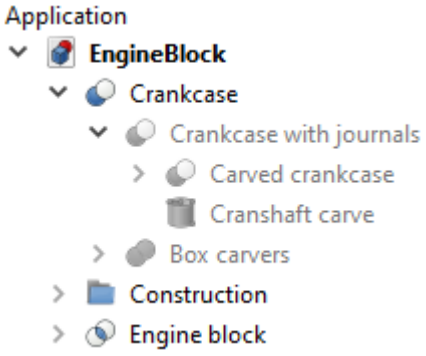


Figure 5.9: FreeCAD tree view containing document objects.

5.6 The Prismatic Spline Sweep

FreeCAD exposes a low-level OpenCascade function named *makePipeShell*, which sweeps a profile about a specified path. If we use the contour as our profile and the polygon blueprints of Section 5.4 as our path, we can use the function to create the correct edge surface.

FreeCAD supports the basis spline, also known as a B-spline, which is a useful tool for smooth curve-fitting of points. By selecting strategic points from a FLOOXS contour, we can represent the same curve with much less information and therefore reduce the computational resource requirement. Representing the contour using a B-spline has the added benefit of allowing a meshing algorithm to model the curved surface with improved accuracy. Fig. 5.10 represents the conversion of the etch profile of Fig. 5.1b into a B-spline.

We wrote a parametric FreeCAD/Python function which creates a shape based on a supplied contour profile, orientation flag, sweep path and shape starting location coordinates. We call the resulting shape a prismatic spline sweep (PSS) because it is a Boolean union of a prism and a sweep of a B-spline profile. The PSS generation process is illustrated in Fig. 5.11. *A* represents the B-spline profile and target path, *B* represents the sweep and *C*, the prism. In this case, the orientation flag has been set to represent positive shape edges.

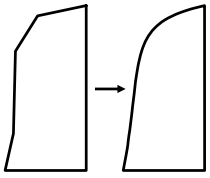


Figure 5.10: Contour to spline conversion.

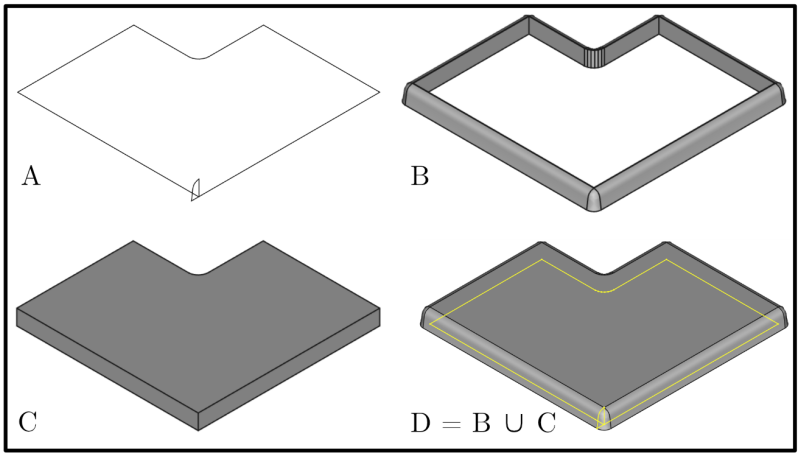


Figure 5.11: Creation of a prismatic spline sweep.

Suppose we set the flag to represent negative shape edges. In that case, the function creates a shape similar to Fig. 5.7, which must be removed from a block of positive material using the Boolean *cut* operation. The Python code for creating a PSS is listed in Appendix C.

5.7 The Connection Between Katana and Parameter Extraction

The three-dimensional model generation module of Katana processes the mask layout, fabrication process data and contour segments to formulate a Python script which, when executed, creates a FreeCAD construction of the circuit. This operation forms part of the larger input-process-output (IPO) model in Fig. 5.12.

The designer can interact with the FreeCAD file through the FreeCAD GUI or Python scripts of their own. They can perform operations specific to their required purposes of their model. Examples of desirable operations include *Boolean fragments* and *FaceSplitter*. These functions are discussed in Section 5.11.

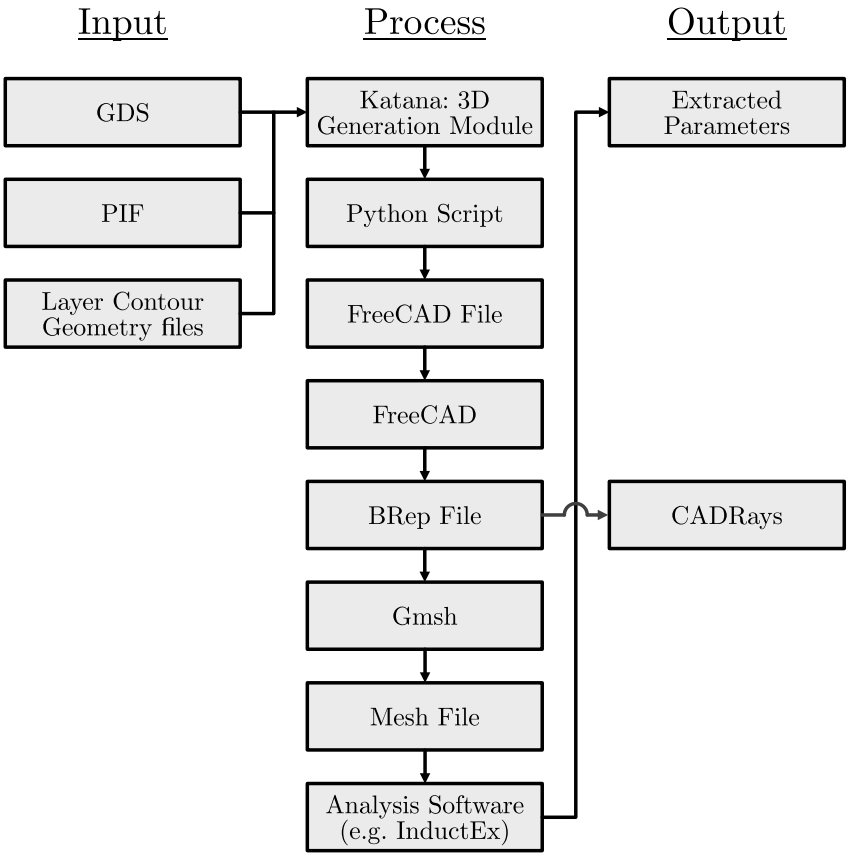


Figure 5.12: IPO model of gate-level extraction using Katana.

Once the designer is satisfied with their model, they can export the entire model or specific layers and subsections into the OpenCascade boundary representation file format (BRep). The designer can then directly import the BRep file into Gmsh for 3D meshing or import it into CADRays [62] if they wish to create a photo-realistic rendering of the model. In Gmsh, the designer can specify precisely how they wish for the program to generate their mesh. They achieve this through their preferred selection of mesh element size, meshing algorithm, element dimensionality and the number of refinement iterations. Mesh customizability is one very crucial benefit of not automating the entire IPO model: There is no one size fits all solution when it comes to meshing.

The designer can use the resulting mesh for FEM simulation tailored to the type of information they wish to gather from the model. In Chapter 6 we discuss examples of where the ColdFlux group have used Katana models to extract valuable parameters.

5.8 The Python Model Script

The Python script generated by Katana instructs the FreeCAD API on how to build the CAD model. Fig. 5.13 details the script contents. The first step towards parametric scripting is to import the FreeCAD modules into the python script. The module exposes the functions required for model generation.

We activate a timer which ends once the script completes so that the designer can gain an understanding of how long subsequent models of a similar scale will take. All layers are processed and saved to a FreeCAD document for in-program viewing.

The script handles layer processing in a bottom-up fashion. There are two primary methods of layer generation. Standard layers are those which consist of a combination of multiple PSS shapes, whereas fill layers consist of a layer block with PSS shapes cut out from it. After a layer is complete, the script saves it as a document object.

Python includes a method called *map* which returns a map object of the results after applying the supplied function to each item of a given iterable. In this case the function is *create_pss* and the iterable is a list of resulting shapes. The *map* function is especially useful for parallel processing, which is discussed in the next section. The "Create PSS Shapes" block of Fig. 5.13 is where we call the map function.

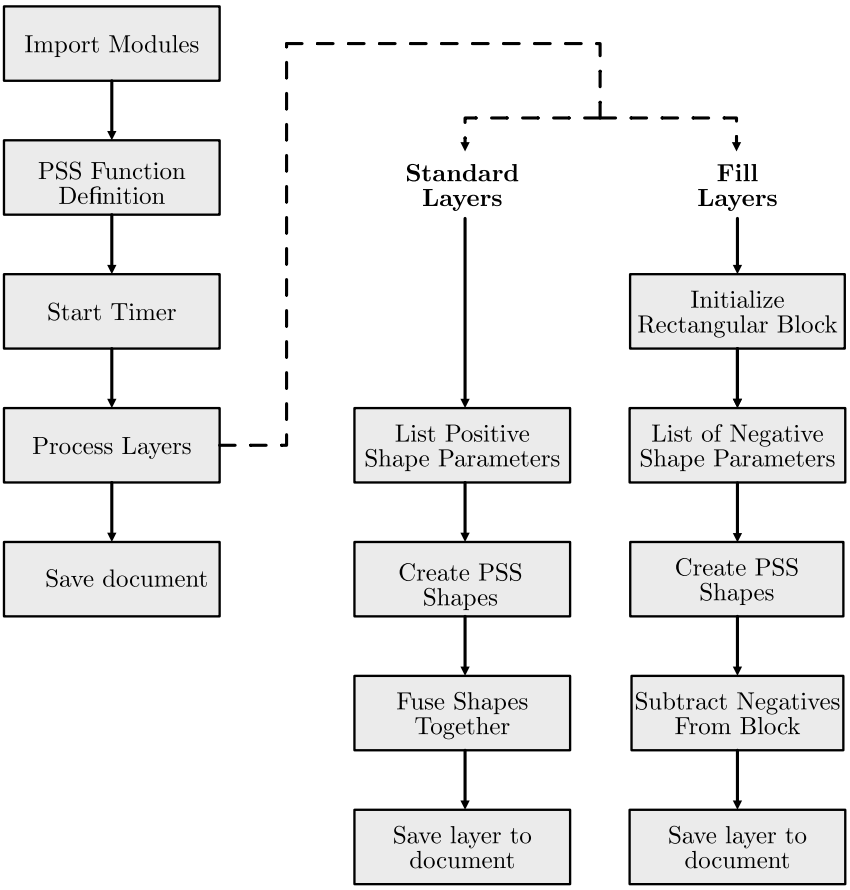


Figure 5.13: Overview of Katana-generated Python script.

5.9 Parallel Processing

Python has less support for parallel processing compared to C++ because of the Global Interpreter Lock (GIL). The GIL is a mutex, which is is a means of ensuring limits on access to a resource are adhered to in an ecosystem where there are many threads of execution. Python supports multithreading. However, because of the GIL, it seldom leads to improved program execution. Python multithreading is usually only used in primarily IO-bound programs [63].

The GIL is a hard limitation of the Python programming language and, where parallel, concurrent solutions are required, it is generally recommended to utilise a different programming language. However, Python does support parallel processing in the form of multiprocessing, which makes use of independent processes (ideally one for each logical core of the system) as opposed to multiple threads within one process. Each Python process is allocated its own Python interpreter and memory. Therefore, the GIL does not need to restrict program execution.

Python includes the *concurrent.futures* module in its standard library. The

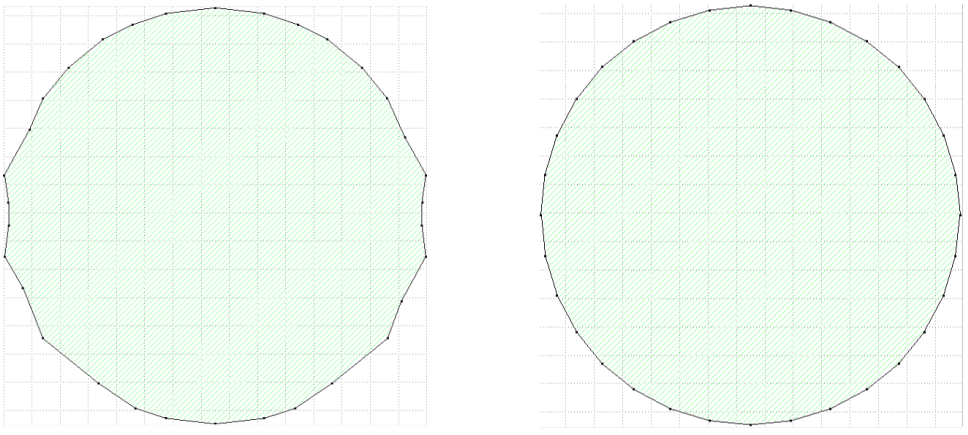
module contains the *pool.map* method, which is a variant of the map method discussed in the previous section. The method automatically assigns list elements from the map to the logical processors of the system the script is running on. The method locks the main program until all processes of the map are complete.

We implemented the *pool.map* method to speed up the shape generation process drastically. Using the system in which we developed Katana, the generation of a single shape such as the one in Fig. 5.11 takes roughly three seconds. The OR2T contains approximately 290 shapes of this kind in the first layer of the OR gate modelled in Chapter 6. By delegating the shape creation task to a server with a large logical processor count, in our case 28 logical cores, it is possible to cut down the computational time of the layer generation from around 15 minutes to half a minute.

The OpenCascade Boolean operations possess a degree of parallel processing. However, because the layer is fused or cut using all of the shapes as the target, we cannot parallelize this code segment. Therefore, it takes up the bulk of time in the script generation process. The results of parallelization in the context of the OR2T gate are discussed in Chapter 6, Section 6.7.

5.10 Circle Healing Algorithm

The GDS file containing the circuit layout may contain malformed circles resulting from unknown design software irregularities. We show such a malformed circle in Fig. 5.14a. These circles cause errors in the PSS creation function and must be repaired before the Python script is generated. We have created an algorithm which identifies malformed circles and replaces them with corrected ones like the one in Fig. 5.14b.



(a) Malformed circle. (b) Healed circle.

Figure 5.14: GDS circle before and after healing.

We run all GDS polygons through a two-factor circle authentication test. First we determine the geometric center, or centroid, of the polygon using the following equations [64]:

$$C_x = \frac{1}{6A}\sum_{i=0}^{n-1}(x_i + x_{i+1})(x_i y_{i+1} - x_{i+1} y_i)\,, \tag{5.1}$$

$$C_y = \frac{1}{6A}\sum_{i=0}^{n-1}(y_i + y_{i+1})(x_i y_{i+1} - x_{i+1} y_i)\,, \tag{5.2}$$

$$A = \frac{1}{2}\sum_{i=0}^{n-1}(x_i y_{i+1} - x_{i+1} y_i)\,. \tag{5.3}$$

Eq. 5.3 represents the formula for the signed area of a polygon.

Once we know the centroid coordinates, we determine the point-to-centroid distance of each point in the polygon and calculate the mean of all of these distances. We compare each point-to-centroid distance against the mean to see if it falls within the specified lower and upper-bounds tolerance band of 10%. This is the first circle authentication factor.

We also calculate the area of a circle of mean point-to-centroid radius and compare that against the area of the actual polygon. We again use the same lower and upper-bounds tolerance method. If both these tests are passed, we consider the tested polygon to be a circle. All polygons which have been flagged as a circle are cleared and repopulated with 32 equidistant points rotated about the centroid.

5.11 Joining Multiple Layers

We possess a FreeCAD model of all layers in the fabrication process once execution of the Python script is complete. Because the designer may wish to model only part of the gate model, these layers are not joined to each other by default. A designer can fuse the required layers by selecting them and running the *Boolean union* operation in FreeCAD's tree view. The resulting combination will still contain surfaces from either side of the layer join. Therefore, before the model can be considered coherent for meshing, we must also run the *Boolean refine* operation. The operation removes any redundant edges and surfaces that remain from the union operation. This operation uses the same underlying OpenCascade code as is used for the creation of a prismatic spline sweep: We remove the redundant join line from the model by calling *removeSplitter* on the part after we join the sweep with the prism.

Chapter 6

Results

6.1 Results Overview

This chapter records the various models created using Katana, using the manual construction methods as well as the automatic generation module. Except for in the thermal simulation, all SiO_2 layers are omitted from the models to reveal the metal layers. All renderings are displayed with the sky plane removed to show more detail of the circuitry.

6.2 ISEC 2019

One of our initial investigations involved determining how process modelling affects parameter extraction of a conventional PTL model. Designers use PTLs to connect circuit elements, allowing for ballistic pulse propagation in single flux quantum (SFQ) circuits.

The investigation compared two two-dimensional cross-sectional models against each other. The one model made use of FLOOXS to generate accurately etched edges, and the other model used the standard approximation of rectangular edges. The investigation found that extracting parameters from the process-modelled geometry yielded a notable difference in results compared to those of the approximated model.

The most significant error value between the two models' parameters was the critical current density, which differed by 8.52%. The experiment was repeated for varying PTL widths. The findings suggest that process modelling is a worthwhile endeavour in the pursuit of obtaining the most accurate results. The full paper, including all of our findings, is contained in Appendix A.

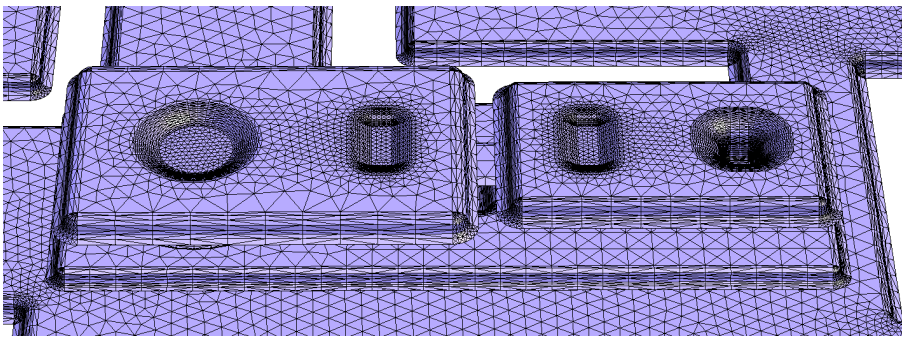


Figure 6.1: Manually modelled JJ with ground plane.

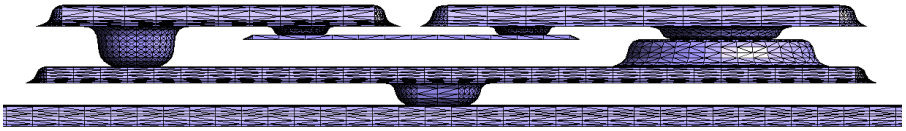


Figure 6.2: Orthographic side view of manually modelled JJ.

6.3 First 3D Josephson Junction

To test the Katana functions created in Chapter 4, we have created a model of a JJ and shunt resistor and then connected it to a ground plane filled with moats. Fig. 6.1 contains an eagle-eye perspective view of the successfully meshed model. We show an orthographic side view of the same model In Fig. 6.2. This model demonstrates that a designer can use Katana to create three-dimensional models from strategically selected cross-sectional FLOOXS simulations. Our findings have enabled us to expand the JJ model into a more detailed circuit element, the JTL.

6.4 Manual Josephson Transmission Line

We manually created a JTL model using the same methodology as the previous Josephson Junction model. We used Katana and FLOOXS to create strategic slices containing the required contours which we rotated, translated, and swept into position. The contours were then manually connected in Gmsh to form surfaces and volumes of sub-models which we merged into the parent construct. After merging the submodules, we then stitched them together using the Gmsh interface's point, line and surface addition tools. Fig. 6.3 contains the reference GDS view of the JTL used to create the process-modelled representation.

The majority of our effort was spent stitching the numerous contours and sub-models together. The Gmsh GUI became substantially less responsive as more points and lines were added into the file. This observation exposed a critical impediment to modelling circuits in this way: Modelling a shape using

discrete points and lines is computationally expensive when compared to using constructive solid geometry-friendly formats such as the BRep file. It took a total of approximately 100 working hours to create the 3D model of the JTL.

Once the model was complete, it was successfully used with InductEx to simulate current density distribution. In the simulation, one JJ was excited as a voltage source. Fig. 6.4 depicts the resulting current density heat map of the manually created JTL model. We show an alternative view which emphasizes the JJ in Fig. 6.5. The colour red represents the highest current density, while blue represents the lowest. One can distinctively see the paths which the current takes, accumulating at the interconnects and around the edges of the moats in the ground and sky planes as well as in the bridging neck of the highest layer (M6).

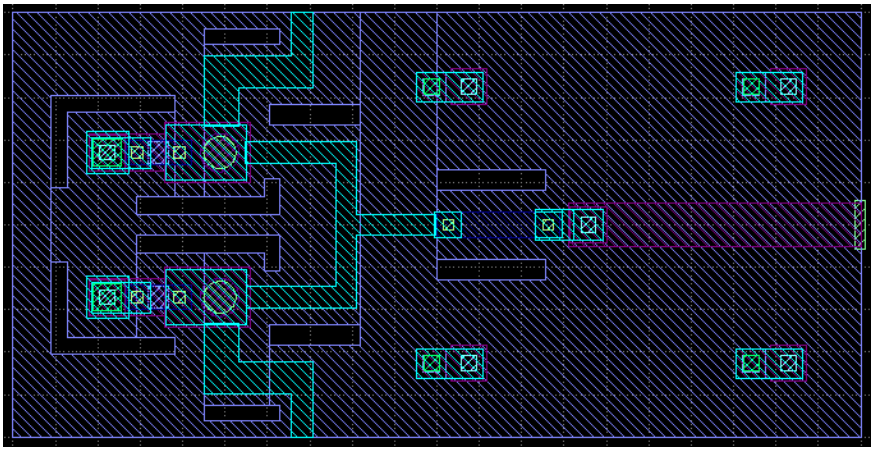


Figure 6.3: GDS view of JTL.

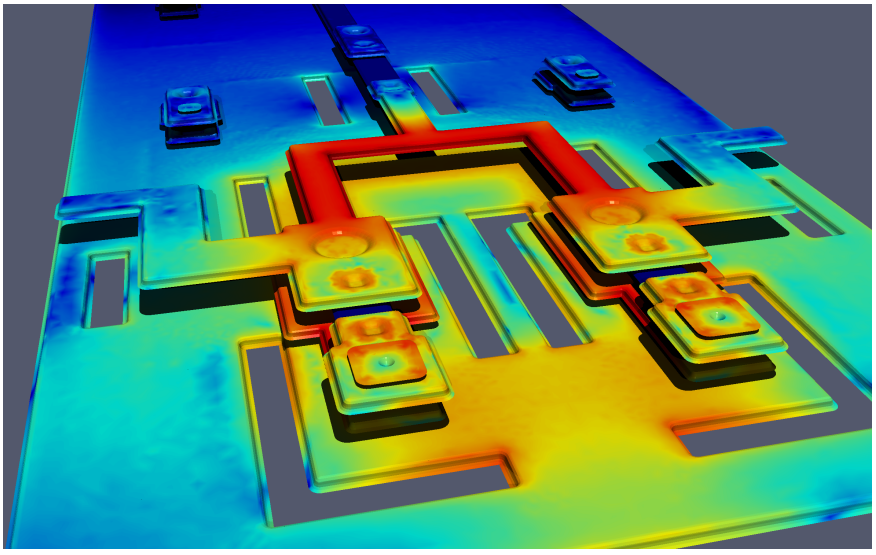


Figure 6.4: Current density heat map of JTL.

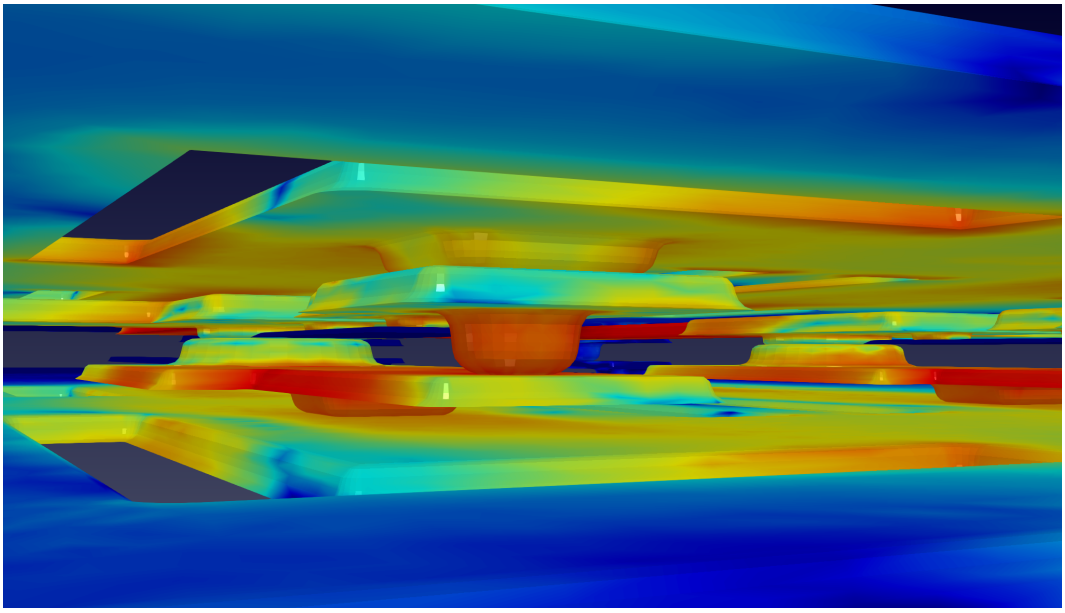


Figure 6.5: Alternative view of manually modelled JTL.

6.5 Automatic Josephson Transmission Line

We took the same JTL GDS file that was used as a reference for manual modelling and supplied it to Katana as an input for the automatic model generator. The model was generated in 49 seconds when using a 6-core processor. In single-thread mode, the model was generated in 1 minute and 13 seconds. The JTL averages between 5 and 10 prismatic spline sweep shapes per layer, so the time gain from using parallelization is not as pronounced as in larger cells. This is because, in the case of a JTL, the shape generation process takes up a smaller portion of the computational time compared to fusing and cutting the layers. When compared to 100 hours, the sub-minute generation time reveals just how valuable automatic model generation can be in the design process. Fig. 6.6 shows the automatically generated model rendered in CADRays.

The model was used as a requisite component of the thermal analysis of the bias resistor on the right-hand side of Fig. 6.3. Fig 6.7 shows the result of a heat transfer simulation of the bias resistor. The simulation was executed in a program called Meltdown, the topic of a separate study [65].

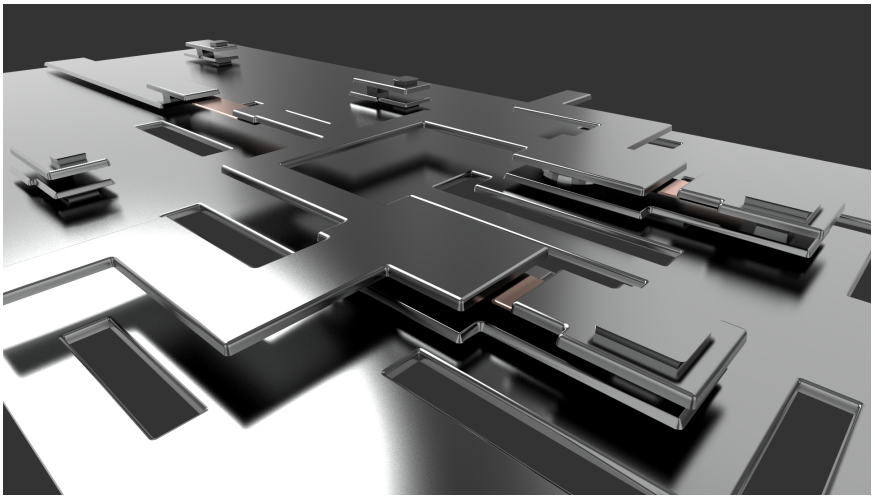


Figure 6.6: CADRays-rendered, automatically generated JTL model.

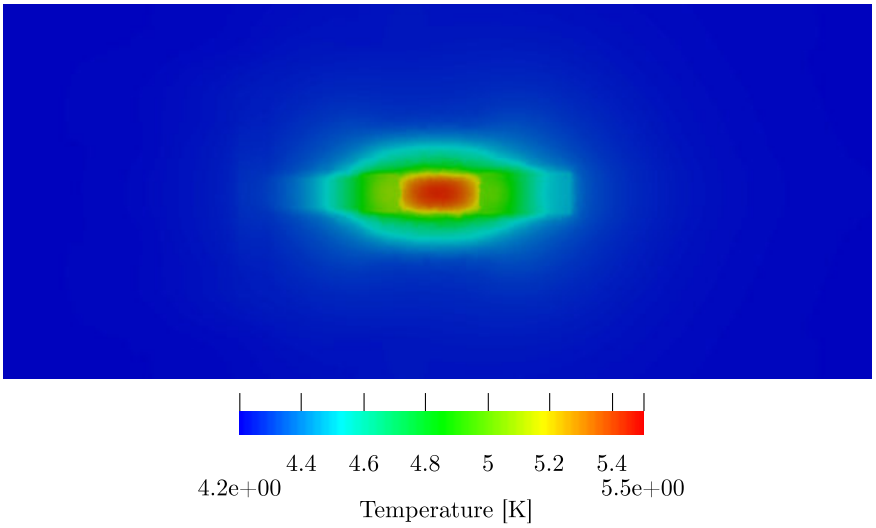


Figure 6.7: Heat transfer from bias resistor into neighbouring material [65].

6.6 SPLITT Cell Model

The SPLITT cell is used to split a single pulse signal line into two synchronous duplicate output pulse signal lines. The cell includes PTL transmitters and receivers and therefore should be connected directly to a PTL [55]. The GDS layout of the cell is shown in Fig. 6.8. We have successfully process-modelled the cell in three-dimensions using Katana. The model is shown in Fig. 6.9 and Fig. 6.10.

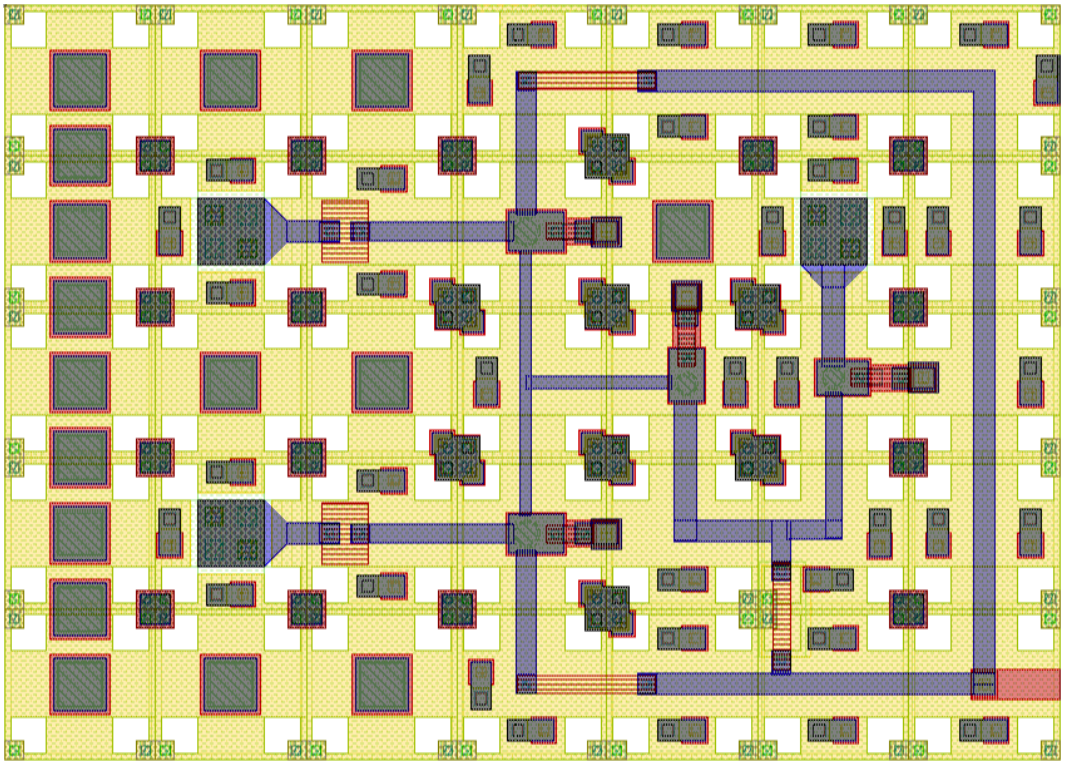


Figure 6.8: GDS view of Splitter cell [55]

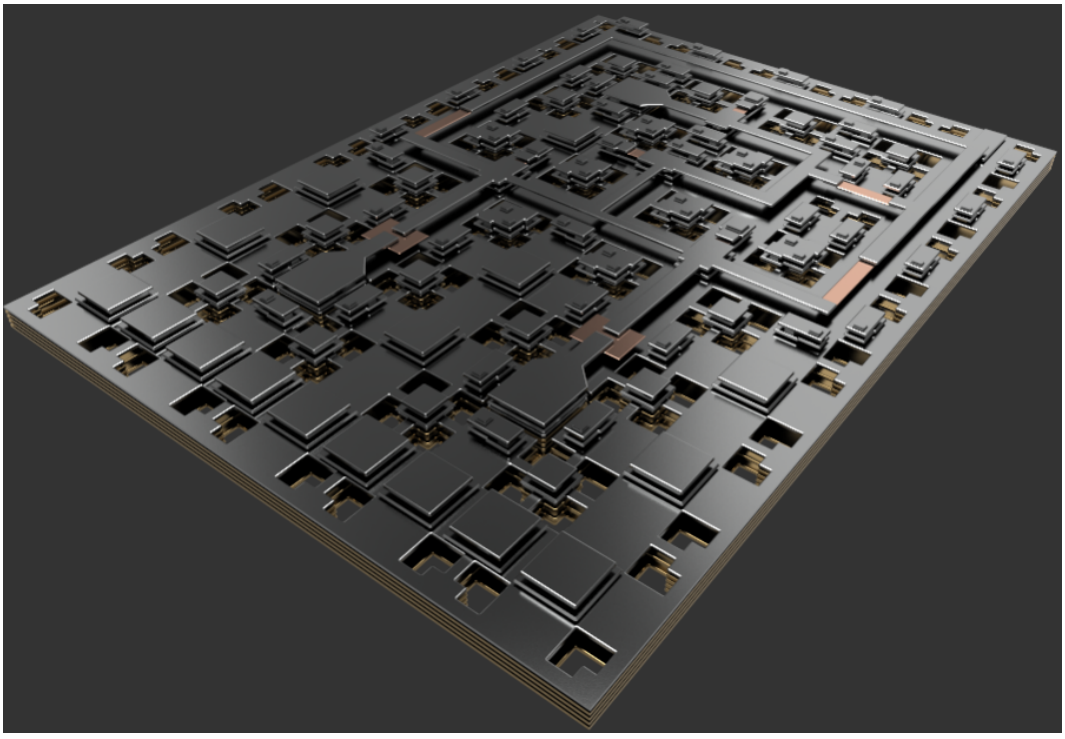


Figure 6.9: Automatically generated splitter cell model.

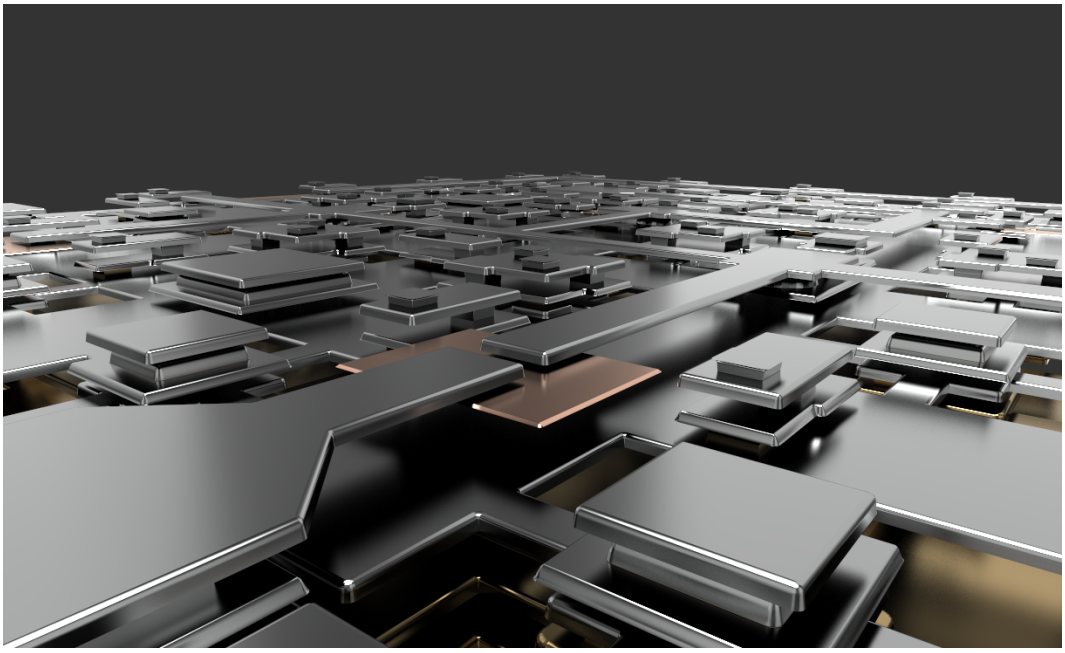


Figure 6.10: Zoomed-in view of splitter cell model.

6.7 OR Gate Model

The OR2T gate has two input channels and one output channel. When either of the input channels provides a pulse to the logic gate ahead of the next clock signal, the gate will transmit a pulse through the output channel at the clock signal. The gate cell includes PTL receivers and transmitters, allowing for direct interface with PTLs [55]. The OR2T logic gate occupies an area of $7000\,\mu m^2$, making it the largest cell in the current generation RSFQ cell library. We show the GDS layout of the cell in Fig. 6.8.

The logic gate creation script contains 6704 prismatic spline sweep parts. Due to the number of parts, in single-thread mode, the shape generation portion alone would take over five hours. Leveraging multiprocessing, we generated the entire logic gate in 17 minutes. The scale of the cell is well beyond the limitations of hand-modelling in Gmsh. In Fig. 6.12, we show a rendering of the complete gate model. We successfully, coherently meshed the model using Gmsh's Delaunay algorithm for three dimensions, satisfying the final requirement of Section 1.3.

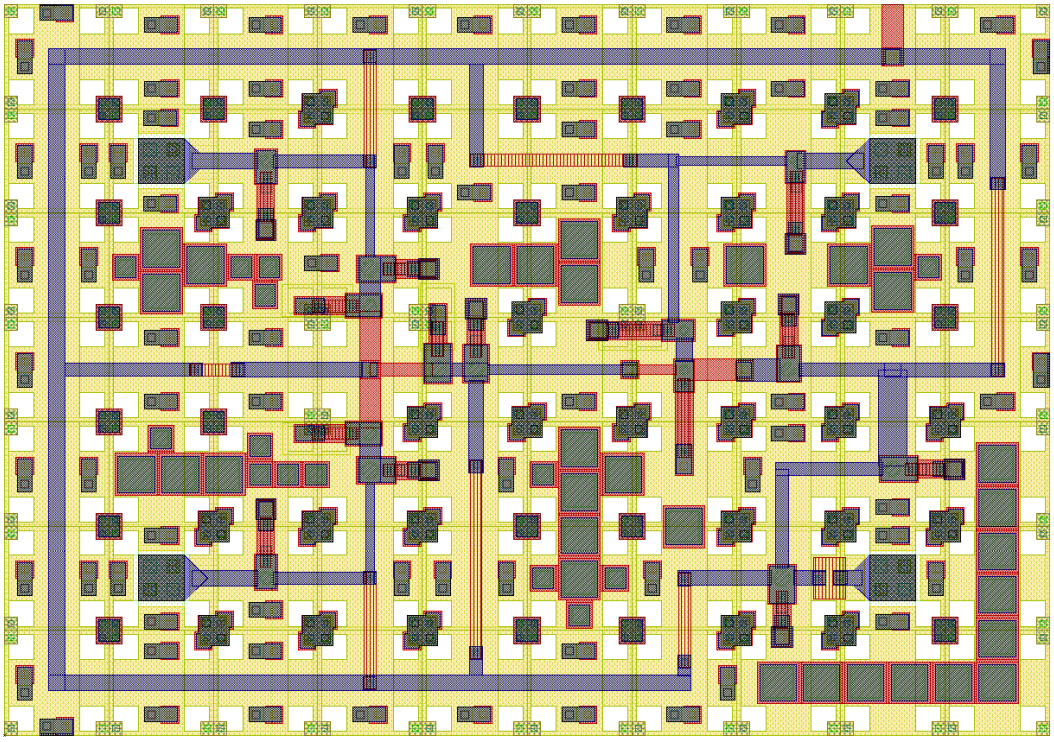


Figure 6.11: GDS view of OR2T cell [55]

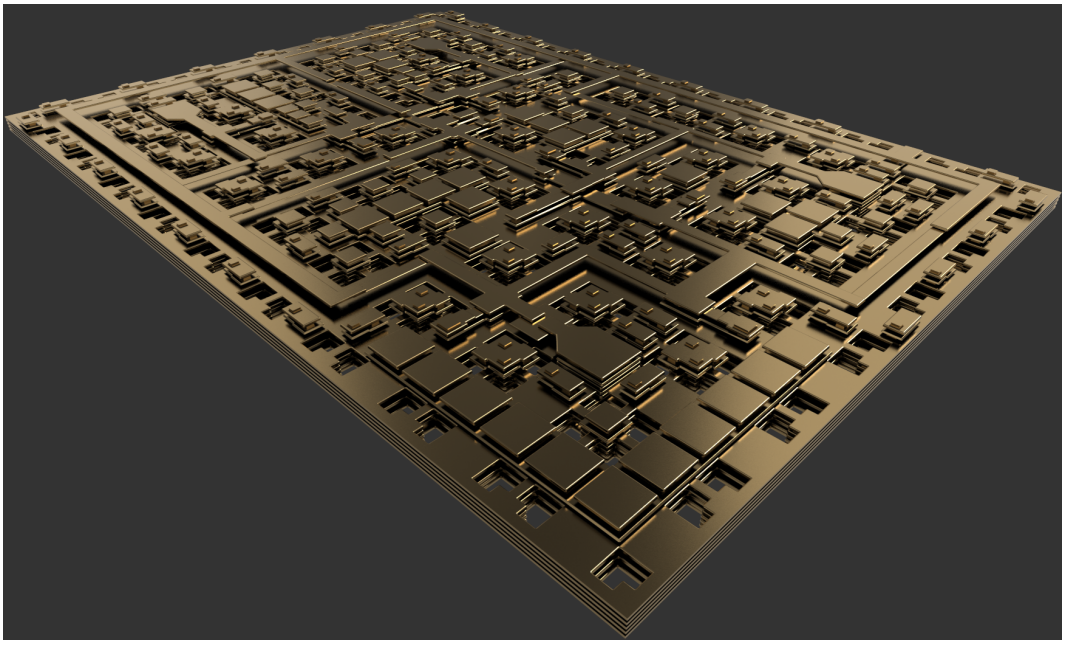


Figure 6.12: CADRays rendering of OR2T logic gate.

6.8 Junction investigation

A three-dimensional Josephson junction model generated with Katana was used by other members of the ColdFlux team in a study to analyze the transient behaviour of the structure. They subdivided the model barrier into small discrete point-junction models and converted the encapsulating metal layers into a tetrahedral mesh. They then converted the discrete point-junctions into a partial element equivalent circuit (PEEC) and used a custom circuit simulator to analyze it. Additionally, the investigators simulated current-voltage characteristics, extracted the equivalent compact SPICE models of the structures and finally simulated the effects of junction barrier defects, external magnetic fields and trapped flux on the dynamics of the JJ structures [66]. Fig. 6.13 reveals the junction geometry created with Katana. Fig. 6.14 demonstrates a simulation of the TCAD junction, with voltage source increased from 0 V to 1 mV in 50 fs. The arrows represent a vector field of current density at different time steps. Each frame corresponds to the time specified by the red dot on the graph.

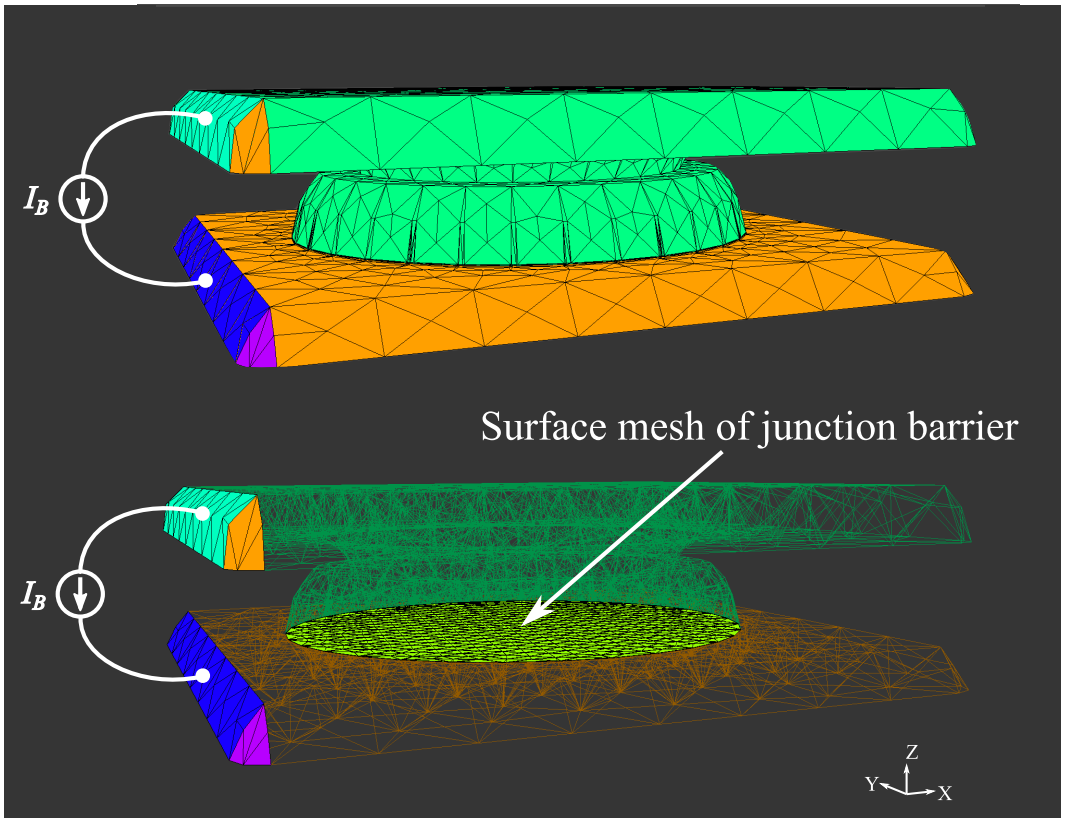


Figure 6.13: Junction geometry for simulation [66].

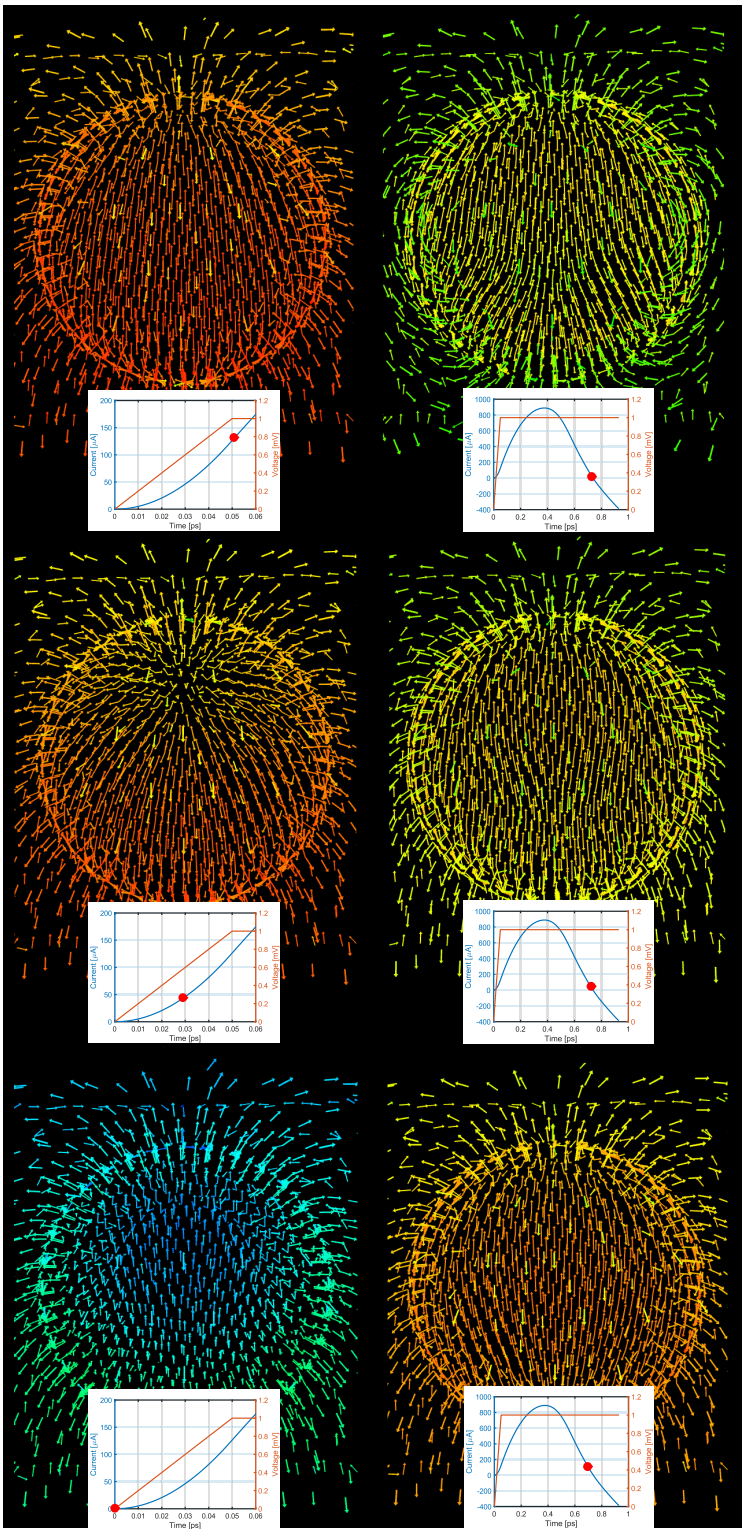


Figure 6.14: Current density simulation of transient voltage event [66].

Chapter 7

Conclusion and Recommendations

7.1 Satisfaction of Objectives

We refer back to the requirements of Section 1.3 to confirm satisfaction of the research project specifications.

7.1.1 Develop An Understanding of the Fabrication Process

Chapter 3 investigates the literature which describes the MITLL SFQ5ee fabrication process. The systematic methods used to fabricate a superconducting IC circuit reveal why circuit geometry looks the way it does and the knowledge gained has subsequently augmented Katana's development. The chapter additionally contains a brief description of the FLOOXS program, as well as the modelling systems utilized in the implementation chapters.

7.1.2 Support Peering Into Circuit Structure and Generating FLOOXS Scripts

Objectives 1 and 2 are satisfied in Chapter 4, which describes the methods used to allow Katana to generate two-dimensional cross-sections of a circuit layout and process simulation input scripts for FLOOXS. In this chapter, we demonstrate a successful cross-section through a PTL model and present a FLOOXS mesh of the PTL edges as initiated by a Katana script.

7.1.3 Implement Tools For Contour Manipulation

Objective 3 is satisfied in Chapter 4 in addition to the previously mentioned objectives. Section 4.6 documents the implementation of SMART-C module for scaling, merging, appending, rotating, translating and making coherent the structures comprised of FLOOXS-generated contour segments. We describe

the modelling process in the same chapter. We verify the functionality module by manually creating the JJ and JTL models presented in Section 6.4 and Section 6.3 respectively.

7.1.4 Automated 3D Fabrication Process Model Generation

The entirety of Chapter 5 is dedicated to explaining the design choices and implementation methods required to build an automatic 3D process model generator. Discussions cover multiprocessing methods, mask layout manipulation, the use of constructive solid geometry principles and programming for the FreeCAD environment. Sections 6.5 to 6.7 successfully demonstrate Katana's capacity to build improved three-dimensional models of RSFQ logic gates. The sections in Chapter 6 also demonstrate Katana actively being used for simulation in other research topics. These models satisfy Objective 4, the cornerstone aim of the research effort.

7.2 Challenges Encountered

Three major obstacles were encountered and overcome during the development of Katana: Scalability limitations of FLOOXS cross-sections, performance issues of geometry file modelling and execution time of single-thread modelling.

7.2.1 FLOOXS Scalability Limitations

Because FLOOXS is still in the process of being adapted for superconducting electronics, there are some areas of the program which require further refinement. One major limitation of FLOOXS is that the simulation time grows exponentially larger when increasing the simulation grid size. The program still struggles with disconnected 2D regions, causing it to crash when a more complex cross-sectional simulation is required. An example of such a case is a multilayer etching and deposition simulation, which is common when one has to model more than just the Josephson Junction. Our solution to this problem was to break the simulation up into smaller parts, extract the contour information, stitch it together again using the methods described in Chapter 4 and re-mesh the whole model.

7.2.2 Performance Issues of Gmsh Geometry File Modelling

During the first iteration of automated model generation, we implemented a script generator which described the model shape surfaces as a combination of points, straight lines and plane surfaces. This method of representation

quickly expanded beyond the capabilities of Gmsh, creating scripts with millions of lines of instructions and freezing the GUI before a designer could even interact with the model. Our solution to this problem was to shift the model creation process outside of Gmsh and use the more compact and accurate method of boundary representation using constructive solid geometry principles in FreeCAD. This transition allowed us to reduce the script size from approximately 1.4 million lines of text for a single layer down to 36,000 for the entire cell. The resulting models of this new method also allowed the mesh to better conform to the correct surface representation of the simulated cell.

7.2.3 Limitation of Single-thread Modelling

Even when using the new method of constructing models, the shape generation step still took long and thus limited the efficiency of any designer who wished to use Katana. To solve this problem, we adjusted the script generation process to utilize the *map* method of Python. The map of shapes to be generated could, in turn, be fed into the parallel processing *pool* module to create shapes of a layer simultaneously, significantly speeding up the model generation process. In the case of the OR2T gate, this optimization reduced execution time by a factor of at least 17.

7.3 Recommendations for Future Work

7.3.1 Automatic Inductex Port Inclusion

One of the primary utilisers of Katana's gate-level models is InductEx, which supports inductance, capacitance and characteristic impedance extraction as well as the calculation of field and current distribution. These are just some of InductEx's features. As a prerequisite to performing these calculations, InductEx requires ports - pairs of terminals which serve as points of entry or exit for electrical energy.

A designer can find the coordinates and specifications of these ports inside the GDS files. Currently, the designer has to specify the ports manually in the generated model. However, we have identified a potential solution to automate this process, provided that we generate a geometry file alongside the Python FreeCAD script. The geometry file should handle the following four-steps:

1. Import the FreeCAD-exported Brep or Step file of the model using the Gmsh *ShapeFromFile* command.

2. Define the plane surfaces which act as ports using the coordinates specified in the GDS file.

3. Use the Gmsh *Volume In BoundingBox* command to identify the volumes which the plane surfaces should belong to automatically.

4. Link the generated plane surfaces to the identified volumes through the *Surface In Volume* function of Gmsh.

5. Assign the plane surfaces to physical groups so that they are identifiable in the final mesh.

Another potential benefit of writing an automatically generated Gmsh geometry script to accompany the FreeCAD script is that Katana could pre-specify to Gmsh a working mesh configuration, including the recommended choice of meshing algorithm and scaling factor. Katana currently generated models using nanometers as the default unit. Therefore, all meshes must scale down by a factor of 10^{-9}.

7.3.2 Journal Papers

The completion of Katana has unlocked a variety of potential journal paper topics. Wherever a designer requires enhanced parameter extraction, they can use Katana models as an input to their simulation software. One potential future paper could, in a similar manner to our 2D PTL investigation, describe the 3D model generation process and then compare extracted parameters of a non-process-modelled gate representation to those of a Katana process-modelled one. If real circuit measurements become available, they could be used as a third reference to determine the degree of accuracy Katana provides over simpler 3D models.

7.3.3 Sinking of Surface Above Vias

Katana's current automatic generation module does not include the upper depressions caused by metal filling into etched vias. A simple and effective way to represent the depressions would be to cut out prismatic spline sweeps from metal layers in areas situated above any identified silicon vias. The spline sweeps would require a once-off extraction of additional contours from a FLOOXS model in a similar manner to those already used for the model.

7.3.4 Full 3D Process Modelling Integration

As the University of Florida is still actively developing FLOOXS, the contours generated by the software may change. If that occurs, we will have to repeat the contour extraction process to create the new contours required by Katana. Additionally, once FLOOXS successfully supports 3D process modelling, we can compare how the 3D features generated by FLOOXS compare to the representation used by Katana and adjust the algorithm if necessary.

7.4 Conclusion

Katana is a program, written in C++ and Python, which creates complex process-accurate 3D models of superconducting RSFQ circuits from mask layout files and fabrication process data. The program has been designed with flexibility in mind and can support other fabrication processes provided a designer has access to the specifications.

Katana creates its models by utilizing constructive solid geometry techniques and leveraging the FreeCAD scripting environment. The CAD engine used by FreeCAD is OpenCASCADE, which is supported by other tools such as Gmsh and CADRays. This relationship enables circuit models created with FreeCAD to be meshed in Gmsh and rendered in CADRays. By using FreeCAD, Katana secures a graphical user interface which allows for manual modification of the generated model if the designer so requires.

During Katana's testing, the program successfully generated RSFQ cell models with nearly 7000 uniquely modelled features. However, the software is likely capable of handling even larger cells given sufficient memory and processing power.

Models generated by Katana have already been used to simulate heat transfer, current distribution and magnetic field distribution. There are plans to expand Katana's modelling capabilities, as well as to investigate new topics through the use of Katana models.

Katana is, along with all of its dependencies, cross-platform and open-source. This transparency ensures that we may benefit from the adaptability and flexibility which open-source software provides. The program's source code is freely available on GitHub [67]. Stellenbosch University also retains a copy of the software.

The work which we have presented has created a solution which makes it possible to quickly, accurately and automatically build complex geometrical models which we can successfully explore with even more complicated tools. This development brings us closer to the end goal of accurately and precisely determining the behaviour of a circuit before physically creating it.

List of References

[1] "Why are There Limits on CPU Speed?" 2000. [Online]. Available: https://computer.howstuffworks.com/question307.htm

[2] S. Brown and Z. Vranesic, "Fundamentals of Digital Logic with VHDL Design," 2005.

[3] C. Mack, "The Multiple Lives of Moore's Law," 2015. [Online]. Available: https://spectrum.ieee.org/semiconductors/processors/the-multiple-lives-of-moores-law

[4] "IARPA Launches Program to Develop a Superconducting Computer," Office of the Director of National Intelligence, Tech. Rep., 2014. [Online]. Available: https://www.iarpa.gov/images/files/programs/c3/C3_press_release.pdf

[5] "IARPA Launches "SuperTools" Program to Develop Superconducting Circuit Design Tools," Office of the Director of National Intelligence, Tech. Rep., 2018. [Online]. Available: https://www.odni.gov/index.php/newsroom/press-releases/item/1848-iarpa-launches-supertools-program-to-develop-superconducting-circuit-design-tools

[6] C. J. Fourie, K. Jackman, M. M. Botha, S. Razmkhah, P. Febvre, C. L. Ayala, Q. Xu, N. Yoshikawa, E. Patrick, M. Law, Y. Wang, M. Annavaram, P. Beerel, S. Gupta, S. Nazarian, and M. Pedram, "ColdFlux Superconducting EDA and TCAD Tools Project: Overview and Progress," *IEEE Transactions on Applied Superconductivity*, vol. 29, no. 5, pp. 1–7, aug 2019.

[7] L. Scheffer, L. Lavagno, and G. Martin, *EDA for IC Implementation, Circuit Design, and Process Technology*. Routledge, 2006.

[8] C. T. Wang, "Niobium Processing - The Metal and its Alloys," 1998. [Online]. Available: https://www.britannica.com/technology/niobium-processing/The-metal-and-its-alloys#ref623232

[9] N. Takeuchi, D. Ozawa, Y. Yamanashi, and N. Yoshikawa, "An Adiabatic Quantum Flux Parametron as an Ultra-low-power Logic Device," *Superconductor Science and Technology*, vol. 26, p. 035010, 01 2013.

[10] S. K. Tolpygo, V. Bolkhovsky, R. Rastogi, S. Zarr, A. L. Day, E. Golden, T. J. Weir, A. Wynn, and L. M. Johnson, "Advanced Fabrication Processes for Superconductor Electronics: Current Status and New Developments," *IEEE Transactions on Applied Superconductivity*, vol. 29, no. 5, pp. 1–13, aug 2019. [Online]. Available: https://ieeexplore.ieee.org/document/8666758/

[11] J. C. Gallop, *SQUIDs, the Josephson Effects and Superconducting Electronics*. CRC Press, 2017.

[12] K. K. Likharev, O. A. Mukhanov, and V. K. Semenov, "SQUID '85-Superconducting Quantum Interference Devices and their Applications Berlin," Tech. Rep., 1985.

[13] M. Kamon, J. K. White, and M. J. Tsuk, "FASTHENRY: A Multipole-Accelerated 3-D Inductance Extraction Program," *IEEE Transactions on Microwave Theory and Techniques*, 1994.

[14] C. J. Fourie, "Inductex Web Page," 2019. [Online]. Available: http://www0.sun.ac.za/ix/?q=home

[15] C. J. Fourie and W. J. Perold, "Simulated Inductance Variations in RSFQ Circuit Structures," *IEEE Transactions on Applied Superconductivity*, vol. 15, no. 2, pp. 300–303, 2005.

[16] K. Jackman and C. J. Fourie, "Tetrahedral Modeling Method for Inductance Extraction of Complex 3-D Superconducting Structures," *IEEE Transactions on Applied Superconductivity*, vol. 26, no. 3, pp. 1–5, 2016.

[17] M. Law, "Main Page - Flooxs," 2017. [Online]. Available: http://www.flooxs.ece.ufl.edu/index.php/Main_Page

[18] C. J. Fourie, "Digital Superconducting Electronics Design Tools - Status and Roadmap," *IEEE Transactions on Applied Superconductivity*, vol. 28, no. 5, pp. 1–12, 2018.

[19] University of Southern California, "Coldflux Website." [Online]. Available: http://coldflux.usc.edu/overview/

[20] J. Al-Khalili, Asim, "VHDL, How It Works," 2013. [Online]. Available: http://users.encs.concordia.ca/~asim/COEN_6501/Lecture_Notes/L9_Slides.pdf

[21] I. C. Society, "IEEE Standard VHDL Language Reference Manual," *IEEE Std 1076-2008 (Revision of IEEE Std 1076-2002)*, pp. 1–640, 2009.

[22] A. B. Kahng, J. Lienig, I. L. Markov, and J. Hu, *VLSI Physical Design: From Graph Partitioning to Timing Closure*. Springer International Publishing, 2011.

[23] L. Schindler, R. van Staden, C. J. Fourie, C. L. Ayala, J. A. Coetzee, T. Tanaka, R. Saito, and N. Yoshikawa, "Standard Cell Layout Synthesis for Row-Based Placement and Routing of RSFQ and AQFP Logic Families," in *2019 IEEE International Superconductive Electronics Conference (ISEC)*, jul 2019, pp. 1–5.

[24] L. W. Nagel and D. Pederson, "SPICE (Simulation Program with Integrated Circuit Emphasis)," EECS Department, University of California, Berkeley, Tech. Rep. UCB/ERL M382, Apr 1973. [Online]. Available: http://www2.eecs.berkeley.edu/Pubs/TechRpts/1973/22871.html

[25] J. A. Delport, K. Jackman, P. l. Roux, and C. J. Fourie, "JoSIM - Superconductor SPICE Simulator," *IEEE Transactions on Applied Superconductivity*, vol. 29, no. 5, pp. 1–5, aug 2019.

[26] R. Martins, N. Lourenço, and N. Horta, *Analog Integrated Circuit Design Automation: Placement, Routing and Parasitic Extraction Techniques*, 1st ed. Springer International Publishing, 2016.

[27] J. L. Semmlow, *Circuits, Signals, and Systems for Bioengineers*, 2005.

[28] "GDSII Graphic Design System User's Operating Manual," 1978. [Online]. Available: http://www.bitsavers.org/pdf/calma/GDS_II_Users_Operating_Manual_Nov78.pdf

[29] L. E. M. Brackenbury, *Design of VLSI Systems - a Practical Introduction*. GBR: Macmillan Press Ltd., 1987.

[30] S. A. Campbell, *Fabrication Engineering at the Micro- and Nanoscale*, 3rd ed. Oxford University Press, 2008.

[31] "Apparatus for CZ Crystal Growth," 2008. [Online]. Available: https://commons.wikimedia.org/wiki/File:Apparatus_for_CZ_crystal_growth.jpg

[32] M. Lincetto, "Silicon Single Crystal," 2008. [Online]. Available: https://commons.wikimedia.org/w/index.php?curid=3896211

[33] Y. A. Cengel and A. J. Ghajar, *Heat and Mass Transfer, Fundamentals & Applications*, 5th ed. McGraw-Hill, 2015.

[34] R. Willardson and E. Weber, *Chemical Mechanical Polishing in Silicon Processing*, 1st ed. Academic Press, 1999.

[35] M. McCoy, "Smooth Success: Chemical Mechanical Polishing is Becoming Indispensable In Chip Manufacturing," 2010. [Online]. Available: https://pubsapp.acs.org:6443/cen/coverstory/88/8828cover3.html

[36] C. Mack, *Fundamental Principles of Optical Lithography*, 1st ed. Wiley, 2008.

[37] V. S. Saji and R. Cook, *Corrosion Protection and Control Using Nanomaterials*, 1st ed. Woodhead Publishing, 2012.

[38] T. Imamura and S. Hasuo, "Characterisation of Nb/AlOx-Al/Nb Josephson Junctions by Anodisation Profiles," *Journal of Applied Physics*, vol. 66, no. 5, pp. 2173–2180, 1989. [Online]. Available: https://doi.org/10.1063/1.344314

[39] "What are Josephson junctions? How do they work?" *Scientific American*, 1997. [Online]. Available: https://www.scientificamerican.com/article/what-are-josephson-juncti/

[40] K. K. Likharev, *Dynamics of Josephson Junctions and Circuits*, 1st ed. Gordon and Breach Publishers, 1986.

[41] S. K. Tolpygo, V. Bolkhovsky, T. J. Weir, A. Wynn, D. E. Oates, L. M. Johnson, and M. A. Gouker, "Advanced fabrication processes for superconducting very large-scale integrated circuits," *IEEE Transactions on Applied Superconductivity*, vol. 26, no. 3, pp. 1–10, 2016.

[42] M. E. Law, E. Patrick, and N. Pokhrel, *FLOOXS Manual*, 1st ed. University of Florida, 2020.

[43] J. R. Shewchuk, "Triangle: A Two-Dimensional Quality Mesh Generator and Delaunay Triangulator," 2005. [Online]. Available: https://www.cs.cmu.edu/~quake/triangle.html

[44] J. F. Remacle and C. Geuzaine, "Gmsh: a Three-dimensional Finite Element Mesh Generator with Built-in Pre-and Post-processing Facilities," *. . . of the 11th International Society of Grid . . .* , 2009. [Online]. Available: https://gmsh.info/doc/preprints/gmsh_paper_preprint.pdf

[45] M. Satran, "Coordinate Systems," 2018. [Online]. Available: https://docs.microsoft.com/en-us/windows/win32/direct3d9/coordinate-systems

[46] A. Johnson, "The Clipper Library," 2014. [Online]. Available: http://www.angusj.com/delphi/clipper/documentation/Docs/{_}Body.htm

[47] "Freecad, the open source 3d parametric modeler," 2019. [Online]. Available: https://www.freecadweb.org

[48] M. K. Agoston, *Computer Graphics and Geometric Modeling: Implementation and Algorithms*. Springer, 2005.

[49] "Constructive solid geometry," 2019. [Online]. Available: https://wiki.freecadweb.org/Constructive_solid_geometry

[50] J. F. de Villiers and H. F. Herbst, "GDSCpp: A C++ Library to Create and Read GDSII Files," 2019. [Online]. Available: https://github.com/judefdiv/gdscpp

[51] Calma, "GDSII Stream Format Manual 1987," 2011. [Online]. Available: http://www.bitsavers.org/pdf/calma/GDS_II_Stream_Format_Manual_6.0_Feb87.pdf

[52] K. Holwerda, "GDSII Format," 1998. [Online]. Available: https://boolean.klaasholwerda.nl/interface/bnf/gdsformat.html

[53] J. F. de Villiers, "Automated Synthesis, Placement and Routing of Large-Scale RSFQ Integrated Circuits, MEng Thesis, Stellenbosch University," 2020.

[54] J. de Villiers, "ViPeR: Verilog Placement and Routing," 2020. [Online]. Available: https://github.com/judefdiv/viper

[55] L. Schindler, "The Development and Characterisation of an RSFQ Cell Library for Layout Synthesis," Ph.D. dissertation, Stellenbosch University, 2020.

[56] J. F. Remacle and C. Geuzaine, "Gmsh Manual," 2009. [Online]. Available: https://gmsh.info/doc/texinfo/gmsh.html

[57] H. F. Herbst, "Katana Pre-release Demonstration," 2019. [Online]. Available: https://www.youtube.com/watch?v=CsfjJpd8BOQ

[58] H. Herbst, P. Le Roux, K. Jackman, and C. Fourie, "Improved Transmission Line Parameter Calculation through TCAD Process Modeling for Superconductor Integrated Circuit Interconnects," in *2019 IEEE International Superconductive Electronics Conference (ISEC) (IEEE ISEC 2019)*, Riverside, USA, jul 2019, p. 5.

[59] H. Goldstein, *Classical Mechanics*. Addison Wesley, 1980.

[60] D. Khattar, *The Pearson Guide to Complete Mathematics for AIEEE*. Pearson, 2010.

[61] A. Johnson, "Polygon Fill types," 2014. [Online]. Available: http://www.angusj.com/delphi/clipper/documentation/Docs/Units/ClipperLib/Types/PolyFillType.htm

[62] OpenCascade, "CADRays: A photorealistic CAD rendering solution." 2016. [Online]. Available: https://old.opencascade.com/content/cadrays

[63] A. Ajitsaria, "What is the Python Global Interpreter Lock," 2017. [Online]. Available: https://realpython.com/python-gil/

[64] P. Bourke, "Notes on Polygons and Meshes," 1997. [Online]. Available: http://paulbourke.net/geometry/polygonmesh/

[65] B. Venter, "Superconductor Tool Development for Heat Propagation, MEng Thesis, Stellenbosch University," 2020.

[66] K. Jackman, P. Febvre, and C. J. Fourie, "Transient analysis of three-dimensional josephson junction structures," 2020, presentation at IEEE Applied Superconductivity Conference, unpublished.

[67] H. F. Herbst, "Katana: Three-dimensional Process-model Generator and TCAD Assistant," 2020. [Online]. Available: https://heinrichherbst.github.io/Katana/

[68] "Improved Transmission Line Parameter Calculation Through TCAD Process Modeling for Superconductor Integrated Circuit Interconnects," *IEEE Transactions on Applied Superconductivity*, vol. 30, no. 7, pp. 1–4, oct 2020.

Appendices

Appendix A

Journal Paper - PTL Expansion [68]

This appendix contains a pre-print of the journal paper, titled *Improved Transmission Line Parameter Calculation through TCAD Process Modeling for Superconductor Integrated Circuit Interconnects*, that was published in the IEEE Transactions on Applied Superconductivity. The paper serves as an expansion to work presented in Riverside, California at ISEC 2019 [58].

Improved Transmission Line Parameter Calculation through TCAD Process Modeling for Superconductor Integrated Circuit Interconnects

Heinrich F. Herbst, Paul le Roux, *Student Member, IEEE,* Kyle Jackman, *Member, IEEE,* and Coenrad J. Fourie, *Senior Member, IEEE*

***Abstract*—The FLOOXS technology CAD (TCAD) process modeling tools developed at the University of Florida have been adapted under the IARPA SuperTools program to support the MIT Lincoln Laboratory SFQ5ee fabrication process. We use FLOOXS to build meshed models of passive transmission lines from superconductor integrated circuit layouts. We have previously developed a numerical solver that extracts transmission line parameters from the meshed model. In this work, we convert a layout slice to FLOOXS inputs, generate 2D meshes of cross-sectional geometries from simulated process steps, and then extract the transmission line parameters from the meshes. Results are shown compared against the results for simplified transmission lines that do not utilize process modeling. The results confirm that process modeling alters the extracted parameter values and suggest that process modeling will become more important as cell density of superconducting electronics increases. We conclude with a discussion on the necessity of process modeling for high-quality parameter extraction.**

***Index Terms*—FLOOXS, passive transmission line, superconductor process modeling, TCAD**

I. Introduction

Technology Computer-Aided Design (TCAD) is well-established for semiconductor processes. Modeling of the Josephson junction fabrication steps for superconductor electronics (SCE) requires the extension of existing TCAD tools. The University of Florida (UFL) is currently undertaking such a task with the active development of the Florida Object-Oriented Device, Process and Reliability Simulator (FLOOXS) [1]. Process modeling is required to determine accurate device geometry, stress profiles, electrical characteristics, and thermal effects, all of which are used to extract high fidelity simulation models of on-chip devices for circuit design and verification. Some of our work towards the ColdFlux project [2] under the IARPA SuperTools program [3] includes the creation of a tool to aid in modeling the process of superconductor electronic fabrication, which uses FLOOXS as the simulation engine.

In this paper we briefly discuss FLOOXS, and then show a two-dimensional cross-section of a passive transmission line (PTL). The cross-section is processed in FLOOXS to generate a two-dimensional mesh which is post-processed before transmission line parameters are extracted with a numerical solver. We then analyze the original simplified model and compare the results of both to evaluate the influence of the new geometry profile. We repeat this process for models of identical layer thickness with varying widths.

The research is based upon work supported by the Office of the Director of National Intelligence (ODNI), Intelligence Advanced Research Projects Activity (IARPA), via the U.S. Army Research Office grant W911NF-17-1-0120. The authors are with Stellenbosch University, Stellenbosch 7600, South Africa (e-mail: 18968708@sun.ac.za; 17500966@sun.ac.za; kjackman@sun.ac.za; coenrad@sun.ac.za).

II. Florida Object-Oriented Process Simulator

The process modeling module of FLOOXS, FLOOPS, simulates how materials are deposited onto an integrated circuit (IC) die. FLOOPS achieves process simulation through the utilization of numerical methods such as the finite element method (FEM) [1] and the level set method (LSM). The LSM tracks the movement of interfaces implicitly. FLOOPS uses the LSM to simulate the deposition and etching processes to construct a two-dimensional mesh of the cross-section being analyzed. A bottom-up approach is followed in the same way that an integrated circuit is fabricated. The user supplies a script to FLOOXS in which information such as deposited material, deposition time and method, and other parameters are defined. Mask-based etching is also handled, but planarization is not yet supported. The simulator then performs functions, such as deposition and etching, in the sequence that they are defined in the script, after which FLOOXS produces a mesh of the required two-dimensional section.

III. Passive Transmission Line Models

A. Description of a PTL

A PTL is a strip of superconducting metal placed between circuit elements to connect them, allowing for ballistic pulse propagation in single flux quantum (SFQ) circuits. It is suitable for modeling with FLOOXS as it is relatively simple, and we can make the simplifiying assumption that the physical properties do not vary along the length of the transmission line. The dimensions of the PTL designed for the MIT-LL SFQ5ee process [4] are shown in Fig. 1. This is the approximate model. Dimensions were chosen to correspond to published results [5].

B. Assumptions

We used the following assumptions for all PTL models: The PTL is infinitely long, perfectly symmetrical and isotropic with no losses in the dielectric. The PTL solver [5] utilizes the

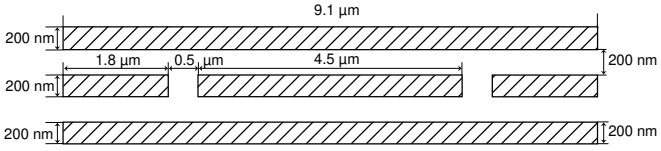


Fig. 1. Passive transmission line cross-section.

Two-Fluid model to determine the complex capacitance $\hat{C}$ and complex inductance $\hat{L}$. No losses are assumed in the dielectric, so that both have only real components. The PTL solver also determines characteristic impedance Z_0, propagation constant γ, and maximum current density I_z of both the simplified and the process modeled PTLs. These parameters are respectively listed in Table I for each model. FLOOXS reveals that the physical profile of the passive transmission line layers differs from the rigid rectangular shape of the approximate model.

IV. Geometrical Modeling

Process modeling is resource intensive. As such, a designer should be able to identify areas of a device or cell layout where process modeling is necessary. Through the combination of process layer information and geometrical descriptions of each layer, it is possible to describe the geometry of a cell fully.

We describe IC layouts in the Calma GDSII stream file format [6]. The GDS files only describe the layout masks, and thus need to be used with process-specific information to derive the physical properties of an integrated circuit die. The designer must also be familiar with the design rules, materials, and layer properties of the fabrication process they are utilizing. The layer definition file (LDF) is a file specification created for use with Inductex [7], [8] that describes the mask and physical information for each layer of the fabrication process. Consequently, each fabrication process has its own associated LDF file.

We have created software (named Katana) which allows the generation of two-dimensional cross-sections through cells. Katana accepts a GDS file, LDF file, and two co-ordinate pairs as an input. It analyzes each layer utilizing a combination of calculated points of interception and the winding number algorithm to solve the point-in-polygon problem to generate the two-dimensional cross-section layer segments. Katana then generates a FLOOXS process-modeling input script based on the cross-section information, which greatly simplifies the

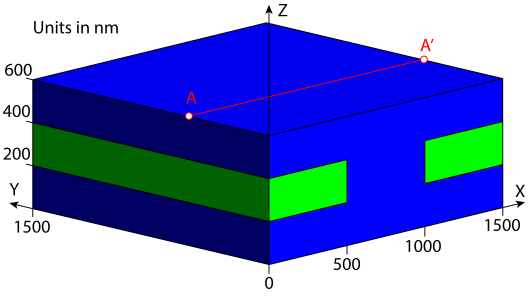


Fig. 2. Two layers of niobium connected by via through silicon dioxide.

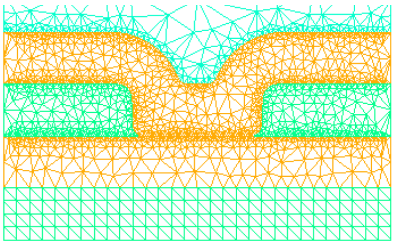


Fig. 3. Sample mesh output from Katana generated FLOOXS script.

modeling process. Gmsh [9] is used for meshing. Katana can also generate a cross-section directly, omitting process modeling. A simple geometrical shape has been generated to illustrate how Katana works. Fig. 2 represents two layers of niobium connected through a layer of silicon dioxide by an etched via through the center. The user calls the cross-section generation module of Katana by using the "Slice" command and specifying the GDS and LDF files and the cross-section coordinates as parameters. The FLOOXS script is then processed by FLOOXS to generate the mesh shown in Fig. 3. Katana allows a designer to peer into multiple relevant areas of a cell quickly to aid in physical model construction.

V. Post-processing

FLOOXS generates an unmodifiable mesh. Therefore, post-processing is required to allow mesh combination and extrusion. Our program converts FLOOXS-output mesh files into Gmsh geometry files (.geo). We optimize the mesh structure by reducing the number of nodes required to represent the mesh file. The program identifies layer boundaries to delimit cross-sectional objects and attributes the respective material layer properties to each such object. The outline of each layer's geometry is retained while the internal mesh structure is removed. The resulting geometry file can be re-meshed and combined with other geometrical data to generate more complex structures. A flow-chart of the simulation process is shown in Fig. 4. The simulation process allows for simplified 3D structures such as the examples in Fig. 5 to be created by extrusion or by a combination of separately oriented two-dimensional segments. Fig. 6 shows an example of the boundary identification and extrusion process of a multiple-layer cross-section of niobium and silicon dioxide. We can combine multiple cross-sections where one is insufficient for

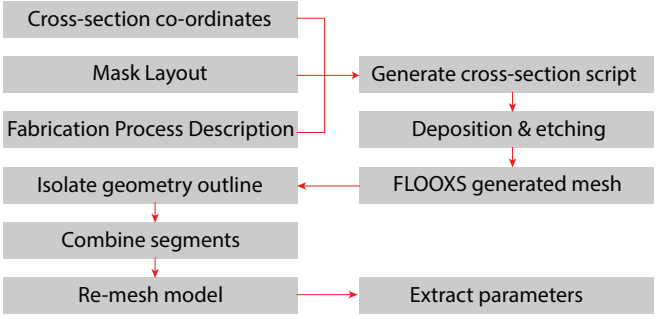


Fig. 4. Flow-chart of simulation process.

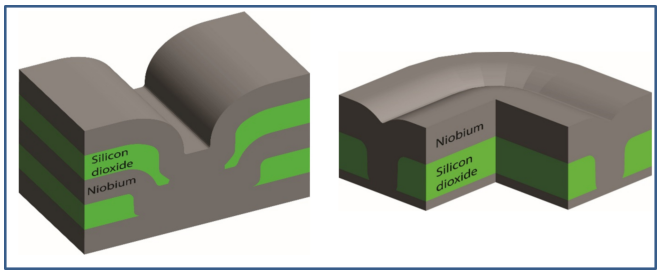


Fig. 5. Example three-dimensional process-modeled structures.

simulating the complexity of a process. We retain control over the mesh to refine the complexity of the mesh in areas of interest.

VI. Results

As a demonstration example, five pairs of PTL models were created with line widths scaled between 0.25 and 1.5 times the widths specified in Fig. 1. The objective is to determine how much of a role process modeling plays with regards to parameter extraction of a PTL. A close-up comparison between the edges of the simplified model and the process model, generated by FLOOXS, can be seen in Fig. 7. In order to reduce resource usage, four separate slices were made with Katana and the resulting FLOOXS meshes were combined programmatically as shown in Fig. 8. The left half of the PTL was modeled and reflected about the PTL's vertical line of symmetry. The geometry of the final model can be seen in Fig 9. The warping and bow effects are emphasized in the zoomed rectangles. As all models assume the length of the passive transmission line is infinite, the 90° bend in Fig. 9 is purely illustrative. Fig. 10 shows the scaled current density of the accurate model. The simulation reveals an increased current density along the tip of the center conductor.

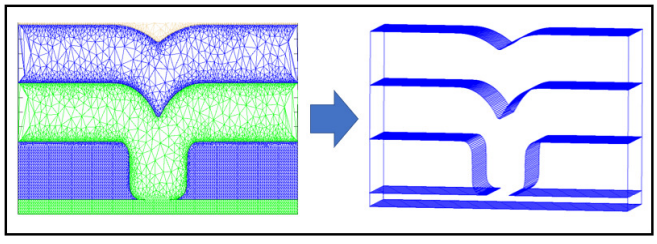


Fig. 6. Boundary identification and extrusion.

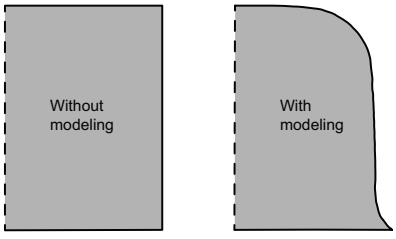


Fig. 7. PTL centre strip edge before and after process modeling.

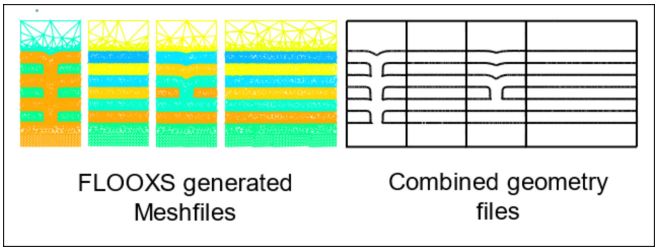


Fig. 8. Illustration of PTL mesh segment conversion and geometry combination.

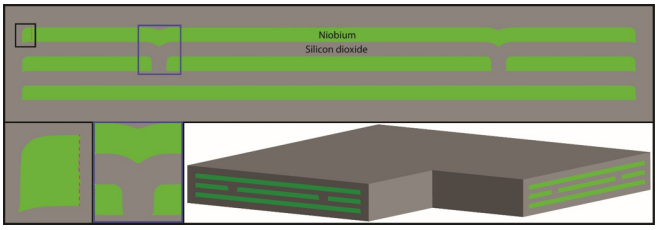


Fig. 9. Final passive transmission line cross-section model.

The extracted parameters for all five PTL models are listed in Table I. The absolute errors in parameters are calculated under the logical assumption that the TCAD-modeled PTL is closer to what the physical parameters would be.

All absolute error values approach zero as the width of the PTL increases. This is due do the edges of the PTL segments becoming less significant as the ratio of etched surface to flat surface decreases. The etched edges of the niobium strip contribute a relatively small amount to the overall parameters in such a wide PTL. In a narrower transmission line, the ratio of curved to flat geometry is higher. Therefore, the error between models using the rectangular approximation and those which accurately model the etches will be more substantial. As discussed previously [5], when localized current density in the corners of the transmission line exceeds the critical current density, superconductivity ceases and current is redistributed. This effect is more pronounced when the conductor has sharp edges. The current crowding effect (CCE) is the change in the distribution of current caused by the tendency of current to

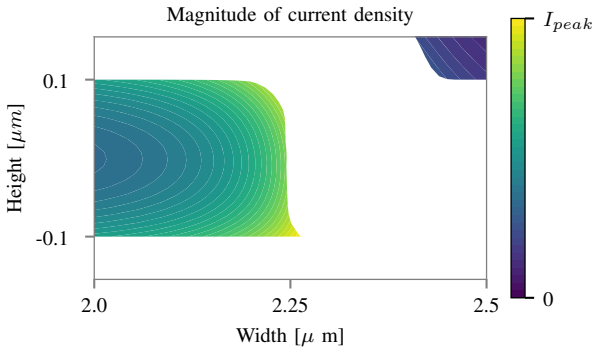


Fig. 10. Current density across centre strip edge of 1.00X Scale process-modeled PTL.

TABLE I
EXTRACTED PER MODEL PTL PARAMETERS.

Scale	Parameter	Approximate Model	Accurate Model	Percentage Error
0.25X	$C\ (nF \cdot m^{-1})$	0.6673	0.6619	-0.81
	$L\ (nH \cdot m^{-1})$	177.40	178.62	0.68
	$Z_0\ (\Omega)$	16.305	16.428	0.75
	γ	683.61	683.19	-0.06
	$I_{peak}\ (A \cdot m^{-2})$	375.68	410.35	8.45
0.50X	$C\ (nF \cdot m^{-1})$	1.064	1.066	0.16
	$L\ (nH \cdot m^{-1})$	100.49	100.79	0.30
	$Z_0\ (\Omega)$	9.717	9.724	0.07
	γ	649.79	651.27	0.23
	$I_{peak}\ (A \cdot m^{-2})$	339.44	369.79	8.21
0.75X	$C\ (nF \cdot m^{-1})$	1.511	1.523	0.79
	$L\ (nH \cdot m^{-1})$	69.72	69.81	0.13
	$Z_0\ (\Omega)$	6.793	6.770	-0.33
	γ	644.88	647.88	0.46
	$I_{peak}\ (A \cdot m^{-2})$	330.86	360.97	9.35
1.00X	$C\ (nF \cdot m^{-1})$	1.964	1.982	0.88
	$L\ (nH \cdot m^{-1})$	53.30	53.26	-0.09
	$Z_0\ (\Omega)$	5.209	5.184	-0.49
	γ	642.92	645.51	0.40
	$I_{peak}\ (A \cdot m^{-2})$	327.70	358.23	8.52
1.25X	$C\ (nF \cdot m^{-1})$	2.422	2.455	0.94
	$L\ (nH \cdot m^{-1})$	43.13	43.00	-0.30
	$Z_0\ (\Omega)$	4.220	4.194	-0.63
	γ	642.12	644.20	0.32
	$I_{peak}\ (A \cdot m^{-2})$	326.49	356.66	8.46
1.50X	$C\ (nF \cdot m^{-1})$	2.880	2.896	0.53
	$L\ (nH \cdot m^{-1})$	36.21	36.20	-0.01
	$Z_0\ (\Omega)$	3.546	3.536	-0.27
	γ	641.67	643.32	0.26
	$I_{peak}\ (A \cdot m^{-2})$	325.97	355.47	8.3

flow through the least resistive path of a circuit [10]. While the CCE was initially observed in semiconductors, the effect applies to superconductors too [11]. It is expected that current crowding at the edges of the transmission line will first cause the current density in the sharp regions to exceed the critical current. However, instead of the strip heating, the current will redistribute so as not to exceed the critical current density, while maintaining superconductivity. For RSFQ transmission lines, which transmit transient SFQ pulses which are not strong enough to break superconductivity, current crowding at the edges of the transmission line can be neglected safely providing the designer follow the design rules specified for the fabrication process. The measured results are more likely to be pertinent when designing higher power transmission lines such as the AC clock/power lines used in AQFP logic. AQFP circuits, such as the one used to demonstrate an AQFP-RSFQ interface [12] operate with clock currents in the mA range, compared to Josephson Junction critical current densities which lie in the order of μA.

VII. CONCLUSION

We have implemented a method to apply process modeling with FLOOXS to superconductor integrated circuits and applied it to the analysis of stripline PTL structures.

The PTL is one of the simplest SCE components. The Josephson junction, for which a process model is under development at the University of Florida [2], is more geometrically complex. Its parameters will deviate more than those of the PTL and process modeling will be substantially more important. Additionally, even slight differences in parameters can have a compounding effect when the circuit element forms part of a larger cell. Without process modeling, high-quality parameter extraction is difficult to achieve. As shown in the semiconductor industry, Technology CAD is a vital stage in the cyclical and iterative process that is circuit design. As the density of superconductor electronics increases, cell sizes must be reduced. Cells will inevitably lie within closer proximity to each other. Because of the need for smaller cell elements, process geometry will play a more significant role in parameter extraction, and therefore TCAD will become a necessity.

Currently, superconductor process modeling is limited to two-dimensional cases. This limitation means that models have to be built with some assumptions made and with reliance on symmetry and one-dimensional isotropic extrusion. Work is currently underway to develop a three-dimensional FLOOXS engine. Until such an engine is a reality, we can automate much of the process of creating a FLOOXS input script and extracting parameters from the resulting mesh with Katana.

REFERENCES

[1] M. Law, "Main Page - Flooxs." [Online]. Available: http://www.flooxs.ece.ufl.edu/index.php/Main_Page

[2] C. J. Fourie, K. Jackman, M. M. Botha, S. Razmkhah, P. Febvre, C. L. Ayala, Q. Xu, N. Yoshikawa, E. Patrick, M. Law, Y. Wang, M. Annavaram, P. Beerel, S. Gupta, S. Nazarian, and M. Pedram, "Coldflux superconducting eda and tcad tools project: Overview and progress," *IEEE Transactions on Applied Superconductivity*, vol. 29, no. 5, pp. 1–7, Aug 2019.

[3] "IARPA SuperTools Program." [Online]. Available: https://www.iarpa.gov/index.php/research-programs/supertools

[4] S. K. Tolpygo, V. Bolkhovsky, R. Rastogi, S. Zarr, A. L. Day, E. Golden, T. J. Weir, A. Wynn, and L. M. Johnson, "Advanced Fabrication Processes for Superconductor Electronics: Current Status and New Developments," *IEEE Transactions on Applied Superconductivity*, vol. 29, no. 5, pp. 1–13, aug 2019. [Online]. Available: https://ieeexplore.ieee.org/document/8666758/

[5] P. le Roux, K. Jackman, J. A. Delport, and C. J. Fourie, "Modeling of Superconducting Passive Transmission Lines," *IEEE Transactions on Applied Superconductivity*, vol. 29, no. 5, pp. 1–5, aug 2019.

[6] I. Steve DiBartolomeo (Artwork Conversion Software, "All About Calma's GDSII Stream File Format." [Online]. Available: https://www.artwork.com/gdsii/gdsii/

[7] C. J. Fourie, "Inductex Web Page," 2019. [Online]. Available: http://www0.sun.ac.za/ix/?q=home

[8] ——, *Inductex User Manual*, 5th ed. Sun Magnetics, 2019. [Online]. Available: http://www0.sun.ac.za/ix/files/misc/Inductex User Manual.pdf

[9] J. F. Remacle and C. Geuzaine, "Gmsh: a three-dimensional finite element mesh generator with built-in pre-and post-processing facilities." [Online]. Available: http://gmsh.info/

[10] S. Liu, G. B. Shan, C. M. Xie, L. S. Wu, and L. Yi, "The modeling of DC current crowding for Through-silicon Via in 3-D IC," in *16th International Conference on Electronic Packaging Technology, ICEPT 2015*. IEEE, aug 2015, pp. 935–938. [Online]. Available: http://ieeexplore.ieee.org/document/7236732/

[11] D. Okimoto, E. Sardella, and R. Zadorosny, "Profile and crowding of currents in mesoscopic superconductors with an array of antidots," *IEEE Transactions on Applied Superconductivity*, vol. 25, no. 3, pp. 1–4, jun 2015. [Online]. Available: http://ieeexplore.ieee.org/document/6967779/

[12] F. China, N. Tsuji, T. Narama, N. Takeuchi, T. Ortlepp, Y. Yamanashi, and N. Yoshikawa, "Demonstration of signal transmission between adiabatic quantum-flux-parametrons and rapid single-flux-quantum circuits using superconductive microstrip lines," *IEEE Transactions on Applied Superconductivity*, vol. 27, no. 4, pp. 1–5, June 2017.

Appendix B

Example Terminal Execution

This appendix holds scripts and terminal outputs from the Katana PTL cross-section example described by Fig. 4.14.

B.1 Terminal Output

```
hein@Heinbuntu:~/Desktop/katana$ ./katana -slice
PTL.gds mitll_sfq5ee_resistance.ldf 1000 -5000 1000 5000
# ------------------------------------------------------------------------ #
#                          _                                               #
#              _________| |_____________________________                  #
#             [_x_x_x_x_   _______________Katana_______________/            #
#                       |_|                                                #
#                                                                          #
#                           Katana Terminal Edition                        #
#        TCAD Tools for the Superconducting Electronics Community          #
#                                                                          #
# FLOOXS is distrubuted and developed separately by University of Florida. #
#                           www.flooxs.ece.ufl.edu                         #
#                                                                          #
#       FLOOXS and Katana development are funded by IARPA SuperTools       #
# ------------------------------------------------------------------------ #

Importing "PTL.gds" into GDSCpp.
Reached end of library.
GDS file successfully imported.
Vertical cross-section
Slicing between [1000, -5000] and [1000, 5000]
Layer [20], XY=[1000,-5000], Distance [0] (BOUNDARY)
Layer [20], XY=[1000,-4550], Distance [450]
Layer [20], XY=[1000,4550], Distance [9550]
Layer [20], XY=[1000,5000], Distance [10000] (BOUNDARY)
Layer [21], XY=[1000,-5000], Distance [0] (BOUNDARY)
```

```
Layer [21], XY=[1000,-3950], Distance [1050]
Layer [21], XY=[1000,-3350], Distance [1650]
Layer [21], XY=[1000,3350], Distance [8350]
Layer [21], XY=[1000,3950], Distance [8950]
Layer [21], XY=[1000,5000], Distance [10000] (BOUNDARY)
Layer [30], XY=[1000,-5000], Distance [0] (BOUNDARY)
Layer [30], XY=[1000,-4550], Distance [450]
Layer [30], XY=[1000,-2750], Distance [2250]
Layer [30], XY=[1000,-2250], Distance [2750]
Layer [30], XY=[1000,2250], Distance [7250]
Layer [30], XY=[1000,2750], Distance [7750]
Layer [30], XY=[1000,4550], Distance [9550]
Layer [30], XY=[1000,5000], Distance [10000] (BOUNDARY)
Layer [31], XY=[1000,-5000], Distance [0] (BOUNDARY)
Layer [31], XY=[1000,-4250], Distance [750]
Layer [31], XY=[1000,-3050], Distance [1950]
Layer [31], XY=[1000,3050], Distance [8050]
Layer [31], XY=[1000,4250], Distance [9250]
Layer [31], XY=[1000,5000], Distance [10000] (BOUNDARY)
Layer [40], XY=[1000,-5000], Distance [0] (BOUNDARY)
Layer [40], XY=[1000,-4550], Distance [450]
Layer [40], XY=[1000,4550], Distance [9550]
Layer [40], XY=[1000,5000], Distance [10000] (BOUNDARY)
Katana Exited.
```

B.2 Generated Gmsh Geofile

The Katana-generated geofile contents are shown below. The file has been abridged to reduce page count.

```
// =============== Layer number [20] ==============
// ========== Layer with identfied x-sec ==========
//------- Block ------
Point(1) = {0, 0, 0, 1.0};
Point(2) = {4.5e-07, 0, 0, 1.0};
Line(1) = {1, 2};
Point(3) = {4.5e-07, 2e-07, 0, 1.0};
Line(2) = {2, 3};
Point(4) = {0, 2e-07, 0, 1.0};
Line(3) = {3, 4};
Line(4) = {4, 1};
Curve Loop(1) = {1, 2, 3, 4};
Plane Surface(1) = {1};
//------- Block ------
Point(5) = {4.5e-07, 0, 0, 1.0};
Point(6) = {9.55e-06, 0, 0, 1.0};
```

```
Line(5) = {5, 6};
Point(7) = {9.55e-06, 2e-07, 0, 1.0};
Line(6) = {6, 7};
Point(8) = {4.5e-07, 2e-07, 0, 1.0};
Line(7) = {7, 8};
Line(8) = {8, 5};
Curve Loop(2) = {5, 6, 7, 8};
Plane Surface(2) = {2};
//------- Block ------
Point(9) = {9.55e-06, 0, 0, 1.0};
Point(10) = {1e-05, 0, 0, 1.0};
Line(9) = {9, 10};
Point(11) = {1e-05, 2e-07, 0, 1.0};
Line(10) = {10, 11};
Point(12) = {9.55e-06, 2e-07, 0, 1.0};
Line(11) = {11, 12};
Line(12) = {12, 9};
Curve Loop(3) = {9, 10, 11, 12};
Plane Surface(3) = {3};

NOTE: LAYERS 21-40 HAVE BEEN REMOVED TO SAVE PAGES
//========== Surface data ===========
Physical Surface("I") = {4, 6, 8, 16, 18, 20};
Color Green {Surface {4, 6, 8, 16, 18, 20};}
Physical Surface("S") = {2, 10, 12, 14, 22};
Color Blue {Surface {2, 10, 12, 14, 22};}
Physical Surface("V") = {1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23};
Color Grey {Surface {1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23};}
```

B.3 Generated FLOOXS Script

The Katana-generated FLOOXS .tcl script is shown below.

```
# ------------------------------------------------------------------------ #
#     FLOOXS PROCESS MODELING SCRIPT Generated by Katana.                  #
#                       heinrichherbst.github.io/Katana                    #
#                         _                                                #
#               _________| |_____________________________                 #
#             [_x_x_x_x_   _______________Katana_____________/             #
#                       |_|                                                #
#                                                                          #
# FLOOXS is distrubuted and developed separately by University of Florida. #
#                           www.flooxs.ece.ufl.edu                         #
#                                                                          #
#     FLOOXS and Katana development funded by IARPA SuperTools Project     #
# ------------------------------------------------------------------------ #
```

```
# Set FLOOXS meshing engine to Gmsh
pdbSet LevelSet gmsh 1

# Configure grid spacing
set spacing 0.05
set etchspace 0.05
set depospace 0.05

# Initialize vertical grid
line x loc=0    spac=$spacing tag=T1
line x loc=1    spac=$spacing tag=T2
line x loc=1.1  spac=$spacing tag=T3

# Initialize horizontal grid
line y loc=0  spac=$spacing tag=S1
line y loc=10 spac=$spacing tag=S2

# Create gas_filled region (reserve space)
region gas xlo=T1 xhi=T2 ylo=S1 yhi=S2

# Create oxide base. (Must deposit onto something)
region oxide xlo=T2 xhi=T3 ylo=S1 yhi=S2

# Initialize model according to above parameters
init

# Define all mask locations.
mask name=layer_20  left=0.45    right=9.55
mask name=layer_21  left=0       right=1.05
mask name=layer_21  left=1.65    right=8.35
mask name=layer_21  left=8.95    right=10
mask name=layer_30  left=0.45    right=2.25
mask name=layer_30  left=2.75    right=7.25
mask name=layer_30  left=7.75    right=9.55
mask name=layer_31  left=0       right=0.75
mask name=layer_31  left=1.95    right=8.05
mask name=layer_31  left=9.25    right=10
mask name=layer_40  left=0.45    right=9.55

# Set-up viewing window.
window row=1 col=2 width=600 height=600

# Plot initial view before deposition and etching.
plot2d grid gas

# Alternate between depositing and etching
```

```
# Layer M2 [20]
deposit metal time=2 spacing=$depospace
etch metal aniso mask=layer_20 time=2.1 spacing=$etchspace
plot2d grid gas

# Layer I2 [21]
deposit oxide time=2 spacing=$depospace
etch oxide aniso mask=layer_21 time=2.1 spacing=$etchspace
plot2d grid gas

# Layer M3 [30]
deposit metal time=2 spacing=$depospace
etch metal aniso mask=layer_30 time=2.1 spacing=$etchspace
plot2d grid gas

# Layer I3 [31]
deposit oxide time=2 spacing=$depospace
etch oxide aniso mask=layer_31 time=2.1 spacing=$etchspace
plot2d grid gas

# Layer M4 [40]
deposit metal time=2 spacing=$depospace
etch metal aniso mask=layer_40 time=2.1 spacing=$etchspace
plot2d grid gas
plot2d grid gas graph=win2

# Below line is FLOOXS input command if executing from same directory.
# source katana_generated.tcl
```

Appendix C

Prismatic Spline Sweep Function

Listing D.1 contains the function which is used to create a prismatic spline sweep. This code forms the foundation of Katana's automated three-dimensional gate model generation script which is discussed in Section 5.8.

Listing C.1: Prismatic spline sweep function.

```
import FreeCAD, Part, time
from FreeCAD import Base

class Ingredient:
    def __init__(self, gp, ct, ps, sb, of):
        self.gp = gp                    # Ground Path
        self.ct = ct                    # Contour coordinates
        self.ps = ps                    # X, Y, Z shape position
                                        # contour orientation:
        self.sb = sb                    # Start Base (True/False)
                                        # Overlap Factor:
        self.of = of                    # Union works better with
                                        # overlapping shapes.

def create_pss(f_ingr):
    # Convert base and contour to FreeCAD vector
    ts = time.perf_counter()
    fc_gp = [FreeCAD.Vector(i) for i in f_ingr.gp]
    fc_ct = [FreeCAD.Vector(i) for i in f_ingr.ct]
    fc_ps = FreeCAD.Vector(f_ingr.ps)
    #create the shape
    wire = Part.makePolygon(fc_gp)
    gwire = Part.Wire(wire)
    last_p = len(fc_ct)-1
    last_p_z = fc_ct[last_p].z
    spline = Part.BSplineCurve(fc_ct)
    if (f_ingr.sb==True):                   # Not inverted
        if ((f_ingr.of>1)and(f_ingr.of<=2)):# Has overlap scale
            offset_lower_pt = FreeCAD.Vector(
            -fc_ct[last_p].x*f_ingr.of, fc_ct[0].y, fc_ct[0].z)
            offset_upper_pt = FreeCAD.Vector(offset_lower_pt.x,
            fc_ct[last_p].y, fc_ct[last_p].z)
```

```
33              offset_ln = Part.LineSegment(spline.StartPoint,
34              offset_lower_pt)
35              vert_ln = Part.LineSegment(offset_lower_pt,
36              offset_upper_pt)
37              horiz_ln = Part.LineSegment(offset_upper_pt,
38              spline.EndPoint)
39              profile = Part.Shape([offset_ln, vert_ln, horiz_ln,
40              spline])
41          elif ( f_ingr.of == 1 ):            # No overlap scale
42              upper_point = FreeCAD.Vector(spline.StartPoint.x,
43              spline.StartPoint.y, spline.EndPoint.z)
44              vertical_ln = Part.LineSegment(spline.StartPoint,
45              upper_point)
46              horiz_ln = Part.LineSegment(upper_point,
47              spline.EndPoint)
48              profile = Part.Shape([vertical_ln, horiz_ln, spline])
49          else:
50              raise Exception(
51              'Overlap factor must be within the range [1, 2]')
52          profile_wire = Part.Wire(profile.Edges)
53          pl = FreeCAD.Placement(FreeCAD.Vector(0,0,0),
54          FreeCAD.Rotation(FreeCAD.Vector(0,0,1), 180))
55          profile_wire.Placement = pl
56      else: # inverted
57          if ((f_ingr.of>1)and(f_ingr.of<=2)): #has overlap scale
58              spline_end = fc_ct[last_p]
59              offset_point = FreeCAD.Vector(
60              (fc_ct[last_p].x*f_ingr.of), fc_ct[last_p].y,
61              fc_ct[last_p].z )
62              offset_upper = Part.LineSegment(spline_end,
63              offset_point)
64              base_point = FreeCAD.Vector(offset_point.x,0,0)
65              vert_ln = Part.LineSegment(offset_point, base_point)
66              horiz_ln = Part.LineSegment(base_point,
67              spline.StartPoint)
68              profile = Part.Shape([spline, offset_upper, vert_ln,
69              horiz_ln])
70          elif (f_ingr.of==1):#no overlap scale
71              lower_point = FreeCAD.Vector(spline.EndPoint.x,
72              spline.EndPoint.y, 0)
73              vert_ln = Part.LineSegment(spline.EndPoint,
74              lower_point)
75              horiz_ln = Part.LineSegment(lower_point,
76              spline.StartPoint)
77              profile = Part.Shape([spline, vert_ln, horiz_ln])
78          else:
79              raise Exception(
80              'Overlap factor must be within the range [1, 2]')
81          profile_wire = Part.Wire(profile.Edges)
82          pl = FreeCAD.Placement(FreeCAD.Vector(
83          -fc_ct[last_p].x,0,0),
84          FreeCAD.Rotation(FreeCAD.Vector(0,0,1), 0))
```

```
85          profile_wire.Placement = pl
86      makeSolid=True
87      isFrenet=True
88      TransitionMode=2 # Rounded edges
89      os = gwire.makePipeShell([profile_wire],makeSolid,isFrenet,
90      TransitionMode)
91      os=os.removeSplitter()
92      if os.isNull()==True:
93          raise Exception(
94          'Prismatic Spline Sweep creation failed:sweep is null.')
95      face = Part.Face(wire)
96      prism = face.extrude(FreeCAD.Vector(0,0,last_p_z))
97      final_pss = os.fuse(prism)
98      if final_pss.isNull()==True:
99          raise Exception(
100         'Prismatic Spline Sweep creation failed at merge.')
101     final_pss = final_pss.removeSplitter()
102     pl = FreeCAD.Placement(fc_ps, FreeCAD.Rotation(
103     FreeCAD.Vector(0,0,1), 0))
104     final_pss.Placement = pl
105     if final_pss.isNull()==True:
106         raise Exception(
107         'Prismatic Spline Sweep failed: return shape is null.')
108     else:
109         te = time.perf_counter() - ts
110         fmt = format(te, '.2f')
111         print('PSS created in '+fmt+' seconds.')
112     return final_pss
```

Appendix D

Katana User Manual

This appendix contains the *Readme.md* file included with the source code of Katana. The file describes how to set up and use Katana.

Katana EDA Process Modeling Assistant

Last updated on 29 October, 2020. Katana is currently at version 1.0T. Commits will be published to the github page: Katana Github

Overview

Katana is an open-source process-modeling assistant & sectioning program. The program is capable of generating process-modelled 3D models of RSFQ cells by leveraging the FreeCAD engine. These models are generally used for improved electrical parameter extraction of superconducting electronic circuits. Katana also allows a designer to peer into GDS files and generate two-dimensional Integrated Circuit (IC) cross-sections with the assistance of layer definition information. Additionally, Katana is capable of automatically generating a FLOOXS process-modeling input script. Katana can combine multiple two-dimensional Gmsh .geo files and convert .msh files back into .geo files (mesh removal).

Changelog

All updates and bugfixes can be found in "Changelog.txt".

Usage

Katana can be executed with input arguments run interactively. If Katana is run with no input arguments, interactive mode is started. The following input arguments may be used:

```
-help
    Print the help information.

-version
    Print the current version of Katana.
```

Katana is primarily used to generate 3D models of RSFQ cells:

```
-3dmodel
    Create a Python script which generates a 3D process-modelled representation
    of a circuit when given the following parameters:
    1 - The GDS mask layout file of specified circuit
    2 - The process information file for the desired fabrication process
    3 - The path to where the contours for the fabrication process are stored
    4 - The path and desired name of the Python script

    Format <Katana> <model command> <gds path> <pif path> <contours path> <output path>

    e.g. ./katana -3dmodel data/JTL.gds data/mitll_sfq5ee.pf data/3DGen
         data/3DGen/JTLModelOutputScript.py
```

Katana is capable of generating cross sections of circuits from mask and process information:

```
    -slice
       Generate a 2D cross-sectional slice through the IC.
       The GDSII format layout file, as well as a layer
       definition file are required as input parameters.
       Katana also requires the x,y co-ordinate pair of
       where to create the slice.

       Format: <Katana> <slice> <gds path> <ldf path> <x1> <y1> <x2> <y2>

       e.g:    ./katana -slice jj.gds mitllsfq5ee.ldf 500 0 500 1000

       Two files are produced; a geometry file and .tcl file.
       The TCL file should be run by FLOOXS to generate a
       cross-section. Both files are saved in the same directory
       as the Katana executable. The geometry file is named
       "cross_section_output.geo". The FLOOXS input script is
       called "katana_generated.tcl".
```

Katana assists in combining multiple cross-sections in order to create a three-dimensional model. All modeling commands make use of the Gmsh .geo file format. In order to define the characteristic length of all points in the .geo file, add either of the following lines to the top of your .geo file:

```
MeshSpac = 1;
cl__1 = 1;
```

Any double can be used in space of the 1. Refer to desired points with that mesh spacing command. Katana currently only stores one characteristic length variable.

```
Point(1) = {1, 1, 1, MeshSpac};
Point(1) = {1, 1, 1, cl__1};
```

The modeling module has the following functionality:

```
-modeling -m
    Merges two specified geometry files in the following manner:
        1: Imports both files into memory.
        2: Performs coherence check on both file data sets.
        3: Simplifies both file data sets.
        4: Merges both data sets together.
        5: Performs coherence check on the merged data.
        6: Simplifies the merged data.
        7: Writes merged data to specified output file.



    Format <Katana> <modeling> <merge command>
           <first file> <second file> <specified output>

    e.g. ./katana -modeling -m data/left.geo data/right.geo
                  -data/combined.geo

-modeling -sa
    (legacy. Replaced with more versatile merge above)
    Simple append: Joins two 2D geo-files. Append the second
    to the first. Support for points and lines only.

    Format: <Katana> <modeling> <simple append> <first .geo>
    <second.geo> <output.geo> <char. len. override>(optional)

    e.g.    ./katana -modeling -sa left.geo right.geo both.geo
    The optional arguments overrides the characteristic length
    of all points. See char. length in Gmsh documentation.

-modeling -t
    Translate entire .geo file. Also perform coherence
    optimization and file simplification.

    Format <Katana> <modeling> <translate command>
    <target file> <delta x> <delta y> <delta z>

    e.g. ./katana -modeling -t data/shape.geo 1000 0 1000

-modeling -r
    Rotate entire .geo file. Also perform coherence optimization
    and file simplification. Requires an origin of rotation and
    rotation angles in degrees per axis.

    Format <Katana> <modeling> <rotate command>
    <target file> <origin x> <origin y> <origin z>
    <theta x> <theta y> <theta z>

    e.g. ./katana -modeling -r data/shape.geo 0 0 0 30 30 30

-modeling -scale
    Scale entire .geo file. Also perform coherence optimization
    and file simplification. Requires a scaling factor. Useful
    for when database units are different to expected.

    Format <Katana> <modeling> <scale command>
    <target file> <output file> <factor>

    e.g. ./katana -modeling -scale data/bigshape.geo
         data/smallshape.geo 1e-2
```

Katana has a Gmsh mesh module capable of mesh volume calculation, as well as conversion of FLOOXS meshes back into a geometry format representation (.geo).

```
    -meshops -s
        This is a legacy function since FLOOXS now supports Gmsh .geo
        contour exporting. This function converts 2D mesh surfaces
        into physical surfaces with the mesh triangles removed.

        Format <Katana> <mesh file operations> <Silver Lining Command>
        <target mesh> <geo output path>

        e.g. ./katana -meshops -s data/f_out.msh data/convert.geo

    -meshops -v
        Calculate all Gmsh .msh \"Physical Volume\" values in mesh file.

        Format <Katana> <mesh file operations> <Volume Command>

        e.g. ./katana -meshops -v data/example.msh"
```

Dependencies

Katana makes use of Boost functions and therefore requires the library present before building.

Katana is built in C++ with CMake and utilizes the clipper library for gds path construction. For your convenience, the relevant source files are included in this repository.

Katana makes use of the Gdscpp library in order to read GDS files. Four your convenience, the relevant source files are included in this repository.

Katana was coded with elements of C++17 and therefore requires a suitable compiler.

Installation on CentOS 7

In order to build Katana easily on CentOS 7, it is recommended to use Software Collections Developer Toolset 8. The toolset allows one to avoid having to build a gcc version greater than 7 from source as this process takes around 4 hours on a modern computer. Once Katana is built you can simply turn the toolset off again.

```
 sudo yum -y install centos-release-scl-rh devtoolset-8 boost-devel git
git clone git@github.com:HeinrichHerbst/Katana.git
cd Katana
wget https://github.com/Kitware/CMake/releases/download/v3.17.0/cmake-3.17.0-Linux-x86_64.tar.gz
tar xf cmake-3.17.0-Linux-x86_64.tar.gz
rm cmake-3.17.0-Linux-x86_64.tar.gz
mkdir release
cd release
scl enable devtoolset-8 bash
../cmake-3.17.0-Linux-x86_64/bin/cmake -DCMAKE_BUILD_TYPE=Release ..
make
exit
```

Installation on Ubuntu 19.10

Same as the CentOS install except you can simply install the required software with the native package manager.

```
 sudo apt install cmake gcc git libboost-all-dev
git clone git@github.com:HeinrichHerbst/Katana.git
cd Katana
mkdir release
cd release
cmake -DCMAKE_BUILD_TYPE=Release ..
make
```

A note on Python virtual environments

Katana uses the FreeCAD Python API to generate 3D models of RSFQ cells. The generated Python scripts have been tested with FreeCAD V0.19, which uses Python 3.8.4. To ensure compatibility, we recommend that you use a virtual environment with this matching version of Python. The "requirements.txt" file contains the requirements of the virtual environment we used to create functional 3D models. See https://youtu.be/Kg1Yvry_Ydk for a guide to using virtual environments in Linux. See https://gist.github.com/HeinrichHerbst/67ac01d84fbd1c8339ec46f0e23d43f4 for our guide to setting up the correct Python venv on Windows using VS Code.